Some approaches to graph fragmentation with application to clustering geo-tagged data

Vu, Ngoc Tuan

Awarding institution:
King's College London

Some approaches to graph fragmentation with application to clustering geo-tagged data

Ngoc Vu

Department of Informatics
King's College London

This dissertation is submitted for the degree of
Doctor of Philosophy

King's College London August 2018

Acknowledgements

I would like to send my wholehearted thanks toward many people without whom it would not be possible to complete the work presented here.

First and foremost, I would like to express my sincerest gratitude to my supervisor, Colin Cooper. My PhD has been an adventure, through many ups and downs, with many joy and *'suffering'*. It is nevertheless an amazing experience and I thank Colin wholeheartedly for his tremendous support, not only in academia (English grammar included) but also many interesting aspects in life, notably: tea, history and Japan. I have learnt and improved a lot through this journey thanks to you.

Cooper-先生, どうもありがとうございます！

Con cảm ơn bố mẹ, vì tất cả. Đây là thành quả của vô vàn công ơn của mẫu hậu và phụ thân. Con cảm ơn ông bà ngoại và nội, vì đây là thành quả của thành quả của các đấng sinh thành của những bậc sinh thành.

Con cảm ơn mọi người trong nhà: Bạch Mi lão bá vì những bài học, ý tưởng và cảm hứng mà hài tử đã học được; bác Hoà, nhờ những chuyến du ngoạn đến các nền văn hoá khác (nôm na là đi chơi) để được mở mang và giảm xì-trét trong những quãng thời gian căng thẳng; và mọi người vì sự quan tâm, động viên và trợ giúp.

To my beloved. My sincerest thanks for your support in life during these years, and also my apologies for many mistakes (some of which I have absolutely no idea what I did wrong). Thank you for sharing many of my interests and for being a motivation for me to improve myself notably the ability to listen to non-stop 'radio' programme.

Cuối cùng, cảm ơn em Ti làm một tiểu đệ ngoan, kiên nhẫn đọc và kiểm tra chính tả cho đại ca, công này thật không hề nhỏ. Chúc em Ti may mắn trong chuyến phiêu lưu của mình.

Abstract

The basic topic of this study is a graph algorithm that *decomposes graphs* which we call *graph fragmentation* algorithms. The aim is to derive fast algorithms to break an input graph into connected subgraphs which we call *fragments*. The original motivation for the research came from data mining geo-tagged photographs from Flickr. Plotting these geo-tags sometimes reveals recognisable patterns of cities, countries or continents made purely of dots. Joining those dots within a limited distance of each other makes a graph many of whose components are cities, attractions, etc. which are densely connected. A question is then: how can one break the graph into *fragments* in such a way that its component structure is preserved?

We give a simple example of a graph fragmentation algorithm. An *active* vertex v is selected. Next, v selects an *active neighbour* u, if any, and v *absorbs* u by making u *inactive* and orienting the edge uv from u to v. If v has no active neighbours then v points to itself, and becomes an *inactive root*. The process continues until all vertices become inactive. In the end, the *fragments* - the objects of interest, are formed by directed paths pointing to the root vertices of the components. Fragmentation algorithms can be varied by altering the *selection* operation. A major difference between the algorithms is how vertices are selected i.e. *probabilistically, deterministically* or *heuristically*. For cycle graphs, we make a formal analysis of the various fragmentation algorithms. We also study a variation in which *edges* are selected instead of vertex, and extend our analysis to circulant graphs.

Many fragmentation algorithms are based on assigning every vertex a unique oriented out-edge, in which case the subgraph obtained consists of uncyclic components. This generalises the subgraph formed by the well-known *random mapping graph* model to which we draw some comparisons.

We next introduce another fragmentation model, the *permutation subgraph* model. The vertices of the graph are permuted and examined in permutation order. Starting from the beginning of the permutation, each vertex points to its first neighbour to the right of it in the permutation, or to itself if no such neighbour exists. Permutation subgraphs are studied in more detail for a wider class of graph models including r-regular graphs, random graphs and infinite random graphs on the integers.

Inspired by the interest in triangles in social networks, we also investigate another variation called *triangle-fragmentation*, in which every vertex points to the neighbour with which has the highest number of common neighbours. Although not a linear time algorithm in general, it seems it might be suitable for decomposing dense graphs. The algorithm is analysed experimentally on planted ℓ partition model and random geometric graphs. It is also evaluated as clustering algorithm on a number of real-world graphs including social networks and graphs formed by geo-tagged photographs taken from Flickr.

Table of contents

List of figures

List of tables

General Notation

■	The end of a partial proof, which contributes to the completion of a full proof.
□	The end of a full proof.
$G(V,E)$	A graph object, consists of the vertex set V and the edge set E.
$V(G)$	The vertex set of a graph G.
$E(G)$	The edge set of a graph G.
$d(v)$	The degree of vertex v.
$\triangle(v)$	Number of triangles associated with a vertex v.
$N(v)$	A neighbourhood of a vertex v, equivalently the set of adjacent vertices of v.
$G(n,p)$	Erdos-Reyni random graph.
C_n^k	k-circulant graph, see Section 6.3 page 95.
$C_n^{2\oplus}$	A subgraph of a C_k^2 graph, see Definition 3 page 96.
K_n	Complete graph with n vertices.

Part I

Introduction to graph fragmentation algorithms

Chapter 1

Research motivation

1.1 Mapping the World's Photos

One of the main motivations behind this work is the research paper by Crandall et al [10] with the title *Mapping the World's Photos*. In this paper, the authors investigate the relationship between geospatial information and visual and textual content of *photographs*. A photograph is often *tagged* with textual content i.e. the owner's description of the picture; and *geo-tagged* i.e. the location where the photograph is taken. There is an enormous number photographs available online, sites like Facebook, Flickr process millions of new pictures everyday. Given the immense amount of information, the photograph resource is a comparable scale corpus with the set of Web pages. Hence, it is useful to think about the analogies between organising photographs and organising Web pages. While successful techniques for analysing Web pages exploit a tight relation between content and structure, existing works that analyse large photo collections have focused primarily either on structure or content [10]. Therefore, the main goal of the paper is to investigate the interplay between *content* i.e. textual tags, image features and *position* i.e. geo-spatial information of large photo collections.

The paper's central thesis is that geospatial information can be directly integrated with visual and textual tag content for organising global-scale photo collections. The main objective of the paper is therefore to study the relation between location and content in large photo-collections. There are many tasks involved in such a study, ranging across different disciplines of computer science. Below we discuss the two tasks which are interesting to us

1. To find the most popular places/location in a large collection of geo-tagged photos.
2. Given the popular photograph locations, to select a visual description or a *representative image* for each specific landmark.

For the *first task* the authors use the *mean-shift* algorithm, which is often used in computer vision for image segmentation and object tracking. By treating geo-tagged photos with their latitude and longitude as points in a two dimensional space, mean-shifting groups points which share the same mode of distribution. Unlike other popular clustering methods such as k-means, mean-shift does not

require a fixed-number of clusters beforehand. Thus, it is suitable for this problem, as the number of clusters is unknown. However it requires a scale-of-observation as input. This is the radius of the searching disc within which every data point will be used to estimate the mode of distribution. The authors used two scales-of-observation: *metropolitan-scale* with the radius 100km; and *landmark-scale* with the radius of 100m. For each scale, the mean-shift procedure is executed to find peaks of the underlying distribution. By counting the number of distinct photographers have taken photos within the respective radius of the searching disc centred at each peak, the authors then produce a popularity ranking of landmarks in each respective location.

For the *second task*, to select a representative image for each popular landmark. The authors search for subsets of photos that are visually very similar and select an image from among the most salient. They posed this as a *graph-problem*. The graph is constructed by considering each photograph as a vertex. For each pair of vertices, an edge is added and given a weight indicating the degree of visual similarities between the pair of photographs. Spectral clustering is then applied to partition the graph into communities so that vertices with each community are highly similar. For each of the clusters, the vertex with highest weighted degree is chosen as the representative image.

In this way, the major attractions of cities around world were identified and tagged. The result for the UK is shown below.

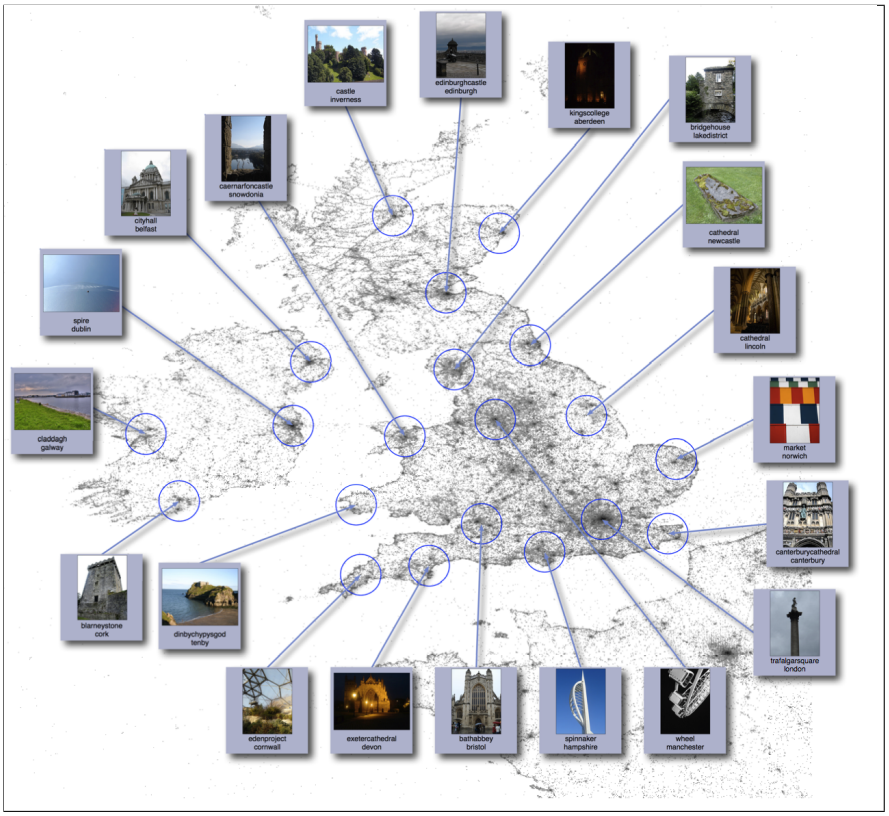


Fig. 1.1 Most photographed landmarks/attractions in the UK as found by Crandall et al [10].

1.2 Replication of the study in *Mapping the World's photos*

The findings in the Mapping the World's photos paper are intriguing. For instance, in London the most popular landmark is Trafalgar Square, followed by the *Tate Modern* as the second-most photographed landmark in London. But Big Ben and London Eye are only ranked as 3^{th} and 4^{th}. Or that the most iconic image of the Trafalgar Square is the Nelson's Column as seen in Figure 1.1, but not the square itself with the beautiful fountains and, perhaps, the pigeons. Therefore, we decided to replicate the study in [10] to have a closer look at the photographs in London.

To collect data, we choose Flickr - a popular photo sharing site, and performed crawling on the users, starting from a group named *London by Londoners*. After two rounds of breadth-first search, the dataset consisted of nearly 11,000 users and 15 millions photos, of which 11 million photos contain textual-tags and 3 million are geo-tagged. For curiosity, we plotted the raw geo-data on the world map, to have an idea of where, presumably, Londoners have travelled and taken photographs. The result is shown in Figure 1.2.

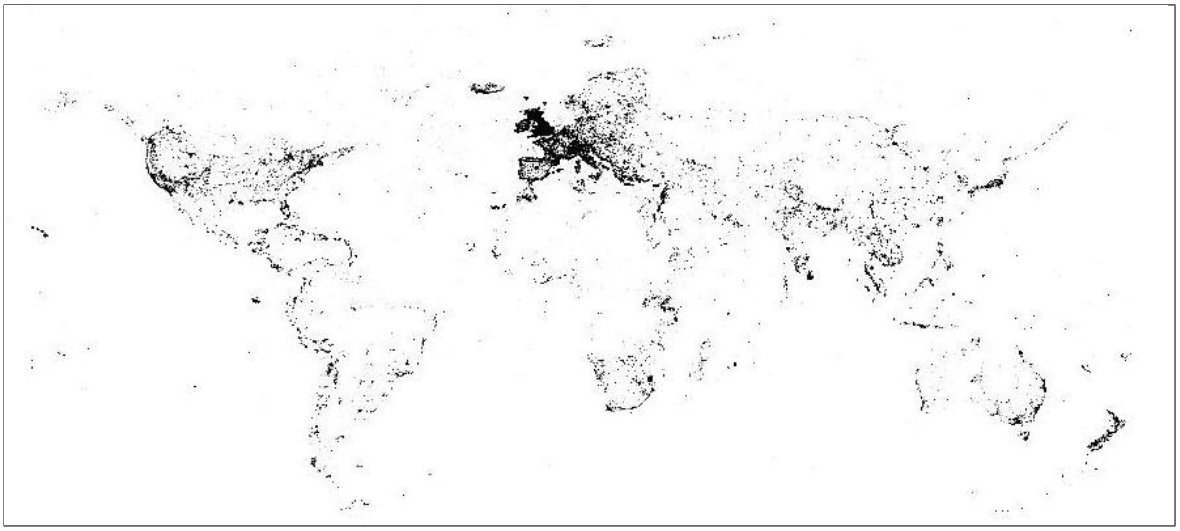


Fig. 1.2 Londoners travel across the world. Remote regions in Canada, Russia, South-America and Africa are significantly less-visible.

Next, we replicated the clustering procedure using *mean-shifting*. The algorithm is discussed in detail in Section 12.3.1, below we give a brief description.

Mean shift is an iterative algorithm for locating the mean of an underlying probability distribution given a set of sample points. The advantage of the mean-shift algorithm is that it requires only a *distance parameter h*. It employs an iterative procedure as follows. For each data point p, we impose a circle with radius h centred at p, the *searching disk*. Using every point located inside the searching disk, *a weighted mean* is calculated. The original p is then *shifted* to the weighted mean. The process then recurses until convergence. In practice this is determined if the shifted distance is less than a constant λ, a parameter of the algorithm.

We provide an illustration for the *mean shift* in Figure 1.3, using a small example from the geo-locations of the photographs.

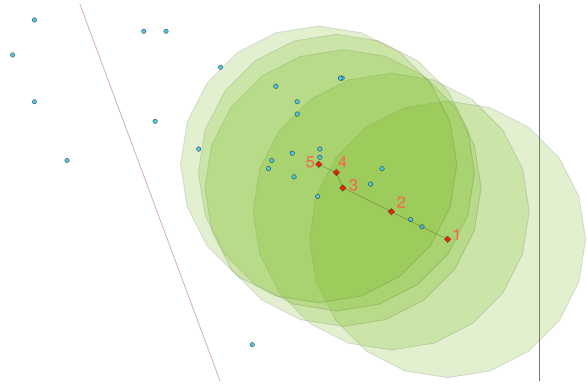
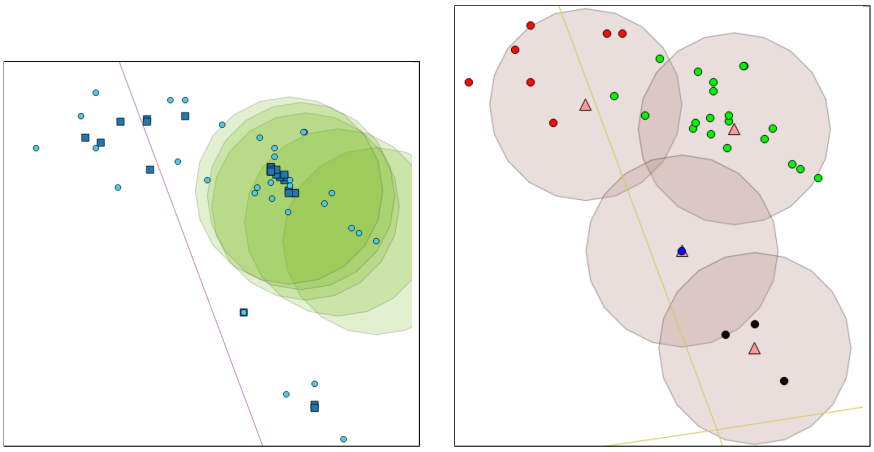


Fig. 1.3 An illustration of the mean-shift algorithm. The *dots* in blue are actual geo-locations of photographs, imposed on world-map with the faint lines indicating streets/footpaths. The *green* circles highlight the searching disk of the queried points. The red, diamond-shaped points are the weighted means (except 1, which is the starting point).

We start from the data point 1. The weighted mean of every point within the disk of 1 is calculated and shown as point 2. The process then recurses with 2 as the queried point. After several iterations it converges at 5 and terminates. The sequence $(1,2,3,4,5)$ highlights the trajectory of the shifted means from start to convergence.



(a) Result obtained from every data point. The green circles are kept from Figure 1.3 for comparison.

(b) Result after post-processing.

Fig. 1.4 The figures above show the results from applying mean-shift to every data point. In figure (a), each *dark-blue square* represents a convergence i.e. a *peak*. As there are many peaks in proximity of each other, a post-processing step can be done by *merging* those within some distance, say h. The result after post-processing is shown in figure (b). In which, points that share the same *merged-peak* (triangle-shaped) are drawn using the same colour. Note that data points are not necessarily located within the searching disk of the final peak, this is a feature of the algorithm.

A problem with *mean-shift* is its high complexity. This is because, in each iteration, the algorithm has to perform a *range search* i.e. searching for points located inside the queried point/mean disk; then calculate a weighted mean. Furthermore, the algorithm is dependant on the parameter λ to determine convergence. In our experiments, we set $\lambda = 10^{-4}$ and noticed for some data points it takes the algorithm a few thousand iterations to reach convergence. Overall, on a small dataset consisting of approximately $4,000$ data points, the experimental running time of mean-shift with radius 10, 25 and 50 metres is 180, 360 and 600 seconds, respectively (for more detail see Section 12.6.2). Thus, given the scale of the dataset, the mean-shift algorithm might not be the best candidate. This motivates us to examine the problem in more detail.

1.2.1 Clustering geographical data as a graph problem

Inspired by the idea of having a searching disk imposed on each data point to search for *neighbours* in its proximity, we can perhaps pose this as a graph problem. More specifically, given the geo dataset we can create a *proximity graph* as follows. Given a parameter d the radius of the searching disk, for every point or vertex v, we connect v to every other points located within the disk centred at v. The result is then a simple, undirected geographical graph, which *captures* necessary information for several operations i.e. range query. Furthermore, plotting the graph on the world map as shown in Figure 1.5 reveals interesting structure of the underlying data.

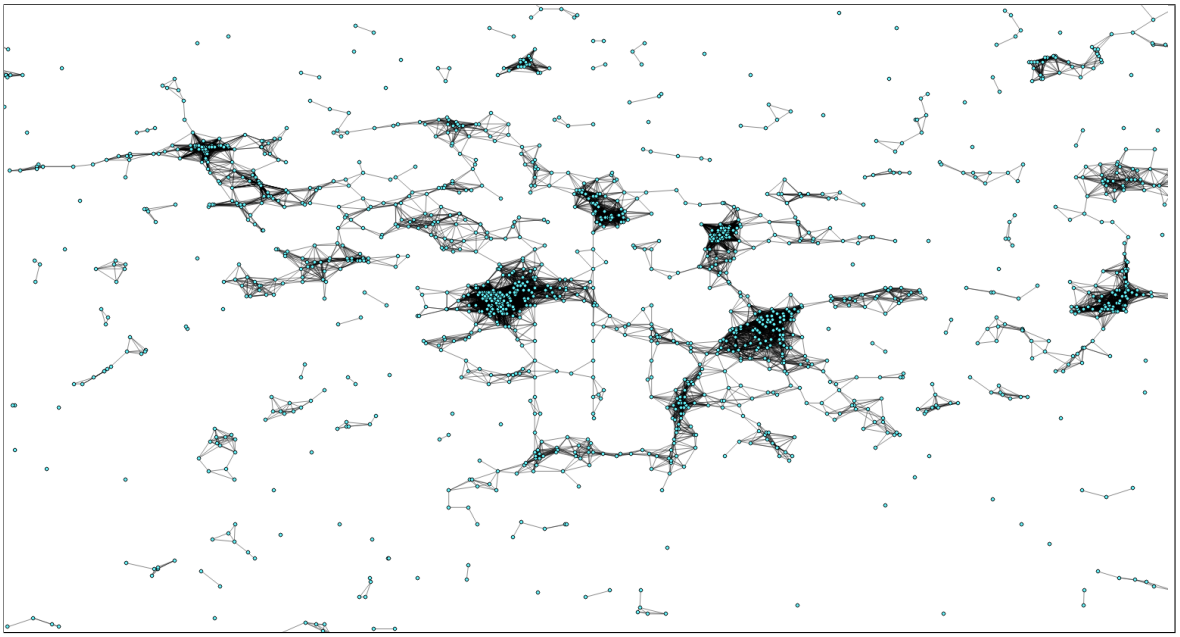


Fig. 1.5 An example of proximity graph. The figure presents a dataset consists of approximately $2,000$ points. For each pair of vertices, we add an edge if the distance between them is at most $d = 100$ metres.

It is seen from Figure 1.5 that there are several *dense regions of points* which are presumably popular locations for taking photographs. When the graph is generated with a correct distance d, these regions are then very densely connected. On the other hand, some of these dense regions are connected by sparse subgraphs. Thus, one can relate this to the graph problem that was posed in task (2) in Section 1.1 that is to identify groups of vertices with dense connections internally and sparser connections between different groups. This is otherwise known as the *community detection* problem.

1.3 Objectives and structure of the study

The motivation for this research is to tackle the problem of clustering geometric/geographical points as a graph problem. That is, given a dataset consists of geo-tags of photographs, a proximity/disk graph can be constructed by connecting pair of vertices of which its distance is at most a chosen constant d. The nature of the dataset is that data-points are distributed with bias at popular landmarks. Consequently, the constructed graph has densely connected components at these locations and is sparser in between. The objective is to partition the graph to reveal these components.

Our main contribution in this research is the introduction, study and analysis of a new graph algorithm, which we call *graph fragmentation* algorithms. The motivation behind this is to study fast methods to decompose an input graph into connected subgraphs which we call *fragments*. The decomposition methods are varied, and can be based on probabilistic, deterministic and heuristic approaches.

To illustrate the algorithms, a simple graph fragmentation procedure is described as follows. An active vertex v is selected uniformly at random from the set of active vertices, v then selects a random active neighbour u, if any. This operation is regarded as the *selection operation*, vertex v is regarded as the *main* vertex and u the neighbour. Next, v *absorbs* u by making u inactive and orienting the edge uv from u to v and the other edges of u are deleted. If v has no active neighbours then v points to itself, and becomes an inactive root.

We provide an overview of the models in Part I. In Chapter 2 we introduce the models and discuss various decomposition methods which are created by altering the selection operation.

In Part II, we make a formal analysis of the various fragmentation algorithms on *cycle graphs* in Chapter 3. In Chapter 6 we study a variation in which *edges* are selected instead of vertex, and extend our analysis to circulant graphs.

Another fragmentation algorithm of interest is *permutation subgraph* fragmentation. The vertices of the graph are permuted and examined in permutation order. Starting from the beginning of the permutation (on the left), each vertex points to one of its neighbours to the right of it in the permutation, or to itself if no such neighbour exists. Permutation subgraphs are studied in more detail for a wider class of graph models including r-regular graphs and random graphs $G(n,p)$ in Chapter 4 and Chapter 5. Many fragmentation algorithms we study are based on assigning every vertex a unique oriented out-edge, in which case the subgraph obtained consists of uncyclic components. This generalises the subgraph formed by *random mappings* to which we draw some comparisons in Chapter 4.

Inspired by the interest in triangles in social networks, we study another variation called *triangle-fragmentation* in Part III. The model is introduced in Chapter 8. In this version, every vertex points to the neighbour with which has the *largest number of common neighbours*. Although not a linear time algorithm in general, it seems suitable for decomposing dense graphs or graphs with high clustering coefficient. The algorithm is analysed experimentally on hidden block models in Chapter 9 and random geometric graphs in Chapter 11.

The triangle-fragmentation approach is evaluated as clustering algorithm on a number of real-world graphs including social networks in Chapter 10 and on graphs formed by geo-tagged photographs taken from Flickr in Chapter 12; thus returning us to our original problem of interest.

Chapter 2

Fragmentation algorithms

Given n data points embedded in two dimensional space, we construct a *proximity* or *disk* graph as follows. Fix a constant d - the *distance parameter*. Consider each data point as a vertex, where each vertex v has coordinates (v_x, v_y). For every pair of vertices v, u if the distance between the pair is at most d, we connect an edge (v,u) i.e. $dist(v,u) \leq d$ where $dist$ is a distant function e.g. Euclidean. The result is a simple, undirected graph G, a proximity/disk graph.

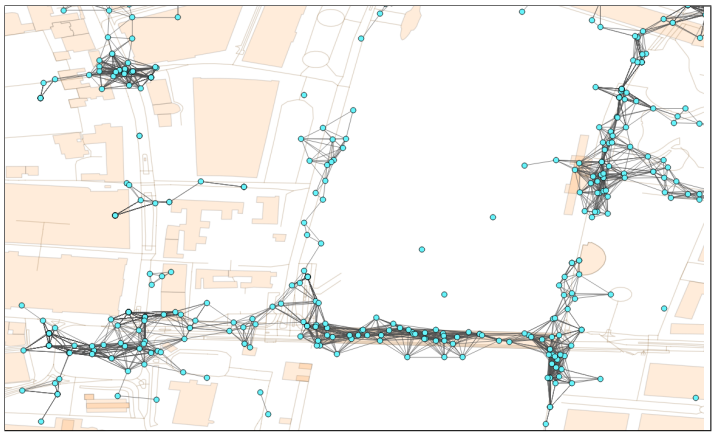


Fig. 2.1 An example of the proximity/disk graph constructed using the geo-locations of photographs. The points are plotted on the world map with roads and buildings, etc. so that reader can have a perspective on the scale of the observation. The figure zooms over the region of Westminster in which the large white gap is the river, to the right is the south bank, etc. The dots on the south bank indicate the location of London Eye. The horizontal line near the bottom of the picture indicates the Westminster bridge and so on.

The nature of the dataset is that there are many *dots* surrounding popular places for taking photographs. Consequently, for a correct value d, these vertices are densely connected. Hence, the objective is to extract these dense components of the graph.

2.1 Fragmentation process

Let $G = (V, E)$ be a simple, undirected graph. For each vertex $v \in V$, v is marked as *active*. The following operation is recursed until options are exhausted

1. Select an *active vertex* v;
2. Select an *active neighbour* u;
3. Let v *absorbs* u, in more detail
   (a) Direct an edge (u, v);
   (b) u is marked as *inactive*.

It is often useful to think of a secondary, directed graph $G' = (V', E')$ to hold the directed edges. More particularly, we copy the vertex set of G to G' i.e. $V' \leftarrow V$ and set $E' \leftarrow \emptyset$. Then each time step 3-(a) is invoked, the new directed edge e.g. (u, v) is added to G'. This edge has a natural orientation with u being the source, and v being the target i.e. $(v \leftarrow u)$ to highlight the event that v pulls or absorbs u. In step 3-(b), u becomes *inactive*. This means we remove u from the vertex set $V = V \setminus \{u\}$ along with its edges $E = E \setminus e(u)$, where $e(u)$ is the incident edges of u. In other words, u is *hidden* and not available for future computation. There may be cases in which the active vertex v has no active neighbour. In this scenario, typically we let v *absorb* itself, add the self-loop (v, v), and regard v as a *root*. Therefore, in each iteration, exactly one vertex becomes *inactive*, hence the algorithm terminates after $|V|$ steps.

The result of the algorithm is held in G', in which each vertex has out-degree 1. Furthermore, each connected component of G' consists of directed paths pointing to the root vertex of that component. To see this, imagine that from any vertex we can always traverse the directed path until we find a self-loop which is the root, the traversal then terminates. We call these components the *fragments*. The fragments are returned as the output of the algorithm. The *first* and *second* steps are the *selection* operations and can be easily altered to create a different variation of the basic fragmentation algorithm. In the following chapters, we will study variations in which selections are based on probabilistic, deterministic and heuristic approaches.

2.1.1 An example of the fragmentation process

We give an example of a simple fragmentation algorithm. Let $G(V, E)$ be a simple, undirected graph. Let $G'(V', E')$ be an empty directed graph. Copy the vertex set of G to G' i.e. $V' \leftarrow V$. Thus, for every vertex $v \in V$ there is a corresponding $v' \in V'$. An active vertex v is selected uniformly at random from the set of active vertices. Next, v selects a random active neighbour u, if any. Next, v absorbs u by making u inactive. We direct the edge (u', v') from u' to v'. If v has no active neighbours then v' points to itself, and becomes an inactive root. The process continues until all vertices become inactive. In

the end, the fragments form a set of components each consisting of directed paths pointing to the root vertex of the component.

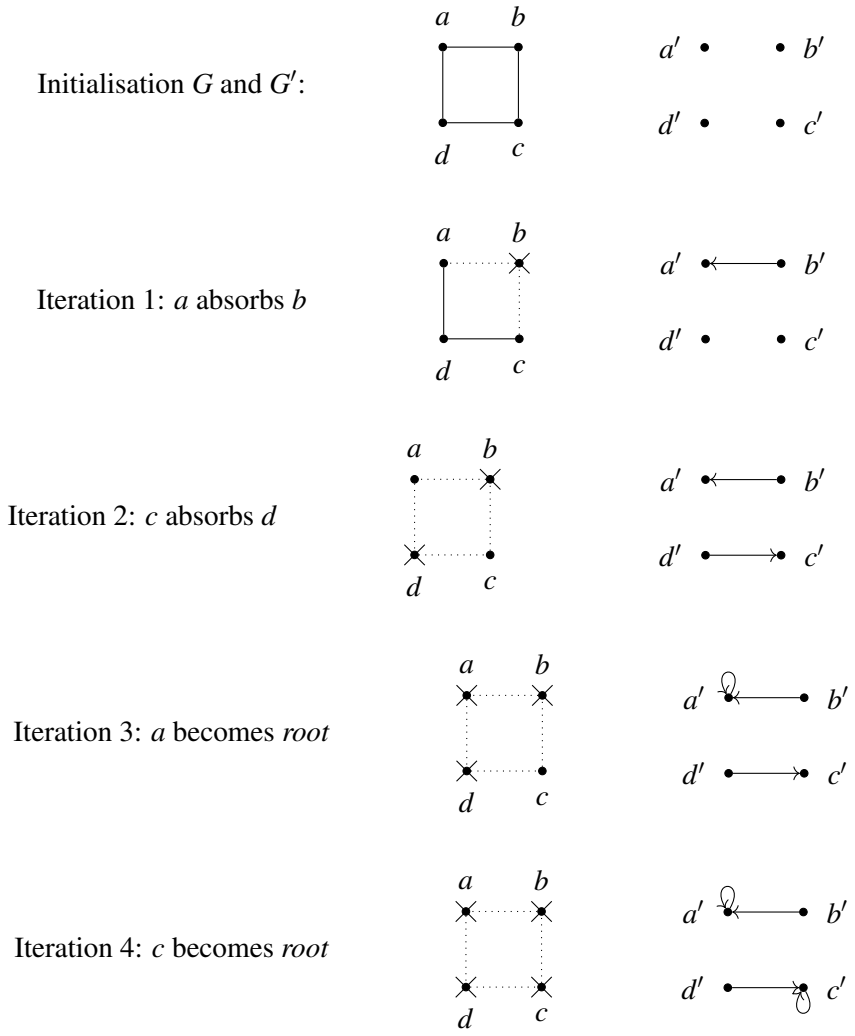


Fig. 2.2 An example of the fragmentation algorithm on an input graph with four vertices $V = \{a,b,c,d\}$. In each iteration the graph G and G' are presented in the left and right figure, respectively. The *crossed out* vertices are *inactive*. The *dotted* lines represent the the incident edges of inactive vertices. The operations are done in the following order

1. a absorbs b; b is inactive.
2. c absorbs d; d is inactive.
3. a has no active neighbour; a attaches *self-loop* and becomes inactive.
4. c has no active neighbour; c attaches *self-loop* and becomes inactive.
5. The algorithm terminates.

2.2 Related models: random mapping graph

The fragmentation algorithm can be viewed as a *mapping* process. In the sense that in each iteration we map a vertex to one of its neighbours or a vertex is mapped to itself. This relationship is represented by the oriented edge. A related graph model is the *random mapping graph* (r.m.g). The r.m.g model is studied in detail by various authors e.g. Bollobas [6], Frieze [22], etc. We give a description of the model below. A more detailed discussion is included later on in Section 4.2.

Let the input graph G be a complete graph i.e. K_n with $V = \{1,2,\ldots,n\}$ hence $|V| = n$. A random mapping directed graph D_f is then constructed as follows. For each vertex i, we map i to one of its neighbour or itself. Equivalently, a directed edge (i, j) with i as the source and j the target is added to D_f. Since the graph is complete and the queried vertex i is also included, the vertex that i maps to is j such that $1 \leq j \leq n$.

Let $\mathscr{D}$ be the set of all possible D_f then $|\mathscr{D}| = n^n$. If for each vertex, its mapping is chosen from a uniform distribution and independently from other vertices, then each graph D_f occurs with equal probability. Each $D_f \in \mathscr{D}$ is a directed graph in which every vertex has out-degree 1. Each component of D_f contains a loop or a cycle. Thus every component is unicyclic. It follows that the number of cycles is also the number of components.

Of many results of r.m.g, one highlights the difference between the model compares to fragmentation (considered in 2.1.1) is that the expected number of components of r.m.g of size n is $\sim \log n$ [6]. Whereas in fragmentation, this number is always 1 as there is always an active pair of adjacent vertices in the graph, up until the last iteration. The set of all possible of mappings of *fragmentation* is a subset of $\mathscr{D}$. In other words, every mapping formed by fragmentation is a *valid* mapping on r.m.g. However, the vice versa is not true. Consider the following mapping examples on a K_7 graph

Vertex :	1	2	3	4	5	6	7
Example (a) - selections :	3	1	4	3	5	7	6
Example (b) - selections:	2	3	4	5	5	5	4

For the *r.m.g model* the above examples can be interpreted as: the vertex at index i (in the first row, which is sorted in increasing order) *maps/points* to the i^{th} vertex in the example row, for example, 1 *points to* 3 in example (a). For the *fragmentation model*, the selections mean: the vertex at index i *is absorbed by* the i^{th} vertex in the example row, for example, 1 *is absorbed by* 3 in example (a). We choose this representation/interpretation to give the resulting edges the same orientation i.e. $1 \rightarrow 3$, equivalently, $e(1,3)$ in both models.

Example (a) is a *valid mapping* of a r.m.g. On the other hand, it is not a valid mapping of *fragmentation* since there are pairs of vertices e.g. $(6,7)$ and $(3,4)$ in which one absorbs the other. This is not possible because any absorbed vertex becomes inactive, thus unable to make any future selection i.e. double-edges are prohibited. Example (b) is a *valid mapping* of a *fragmentation*, it is also valid on r.m.g and both models produce the same resulting graph (see the figure below).

In *r.m.g* each vertex's mapping can be chosen independently from other vertices. Whereas in *fragmentation* the subsequent mappings are dependant on the results of the selections made in previous iterations. In other words, the construction of a *r.m.g* can be thought of as a parallel process; while the *fragmentation* is an iterative process.

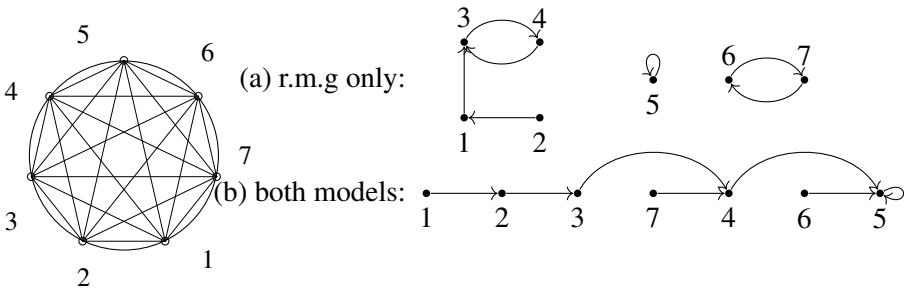


Fig. 2.3 A comparison of the resulting graphs of r.m.g and the fragmentation considered in example (a) and (b) on a K_7 graph. The previous table shows the mappings selected by the vertices. For r.m.g, the interpretation is that the vertex at index [i] *points to* the vertex in the respective index in r.m.g row. For fragmentation, the vertex at index [i] is *absorbed* by the vertex indicated in the row below in the table.

For fragmentation in example (b), the order of operations is as follows. The selections are generated by following the process introduced in Section 2.1:

1. 2 is selected: 2 absorbs 1, 1 is *inactive*;
2. 3 is selected: 3 absorbs 2, 2 is *inactive*;
3. 4 is selected: 4 absorbs 3, 3 is *inactive*;
4. 4 is selected: 4 absorbs 7, 7 is *inactive*;
5. 5 is selected: 5 absorbs 4, 4 is *inactive*;
6. 5 is selected: 5 absorbs 6, 6 is *inactive*;
7. 5 becomes root, 5 is *inactive*.

2.3 Variations of fragmentation

Recall the core procedure of the fragmentation algorithm:

$$\begin{array}{ll} \textit{Vertex selection} & \begin{cases} 1.\ \text{Select an } \textit{active vertex } v; \\ 2.\ \text{Select an } \textit{active neighbour } u. \text{ If there is none, } u = v; \end{cases} \\ \textit{Graph adjustment} & \begin{cases} 3.\ v \textit{ absorbs } u; \ u \text{ is inactive.} \end{cases} \end{array}$$

Step *one* or *two* can be easily altered to create new variations of the basic fragmentation. The *graph adjustment* can also be modified accordingly if required. For example, consider the following modification to the selection of neighbours:

2. Select *all* active neighbours of v.

Then the third step needs to be adjusted:

3. v absorbs all neighbours which become inactive.

Let $N(v)$ be the set of neighbours of vertex v. We have a variation of the basic fragmentation:

Algorithm 1: Fragmentation: a vertex absorbs all neighbours **I.a**

```
while There remains an active vertex do
    Select an active vertex v at random if N(v) is empty then
        v is a root
    else
        v absorbs N(v)
    end
end
```

Let $G = G(n, p)$ a simple, undirected random graph be the input. We are interested in the expected number of components or fragments. Algorithm 1 proceeds on G as follows. Suppose we start with no edge. In each iteration, we pick an *unvisited vertex* v. For each remaining *available to connect* vertex u, the edge (v,u) is connected with probability p. Let $N(v)$ be the set of vertices which are successfully connected to v. Then v absorbs $N(v)$ and forms a new fragment, which is a star graph with every neighbours pointing to v. Thus, all vertices in $N(v)$ are now *inactive* and not available to connect. Further, although v is still *active*, it is not available to connect. In following iterations, if v is visited again then it becomes a root. Else, we select an unvisited vertex and the algorithm recurses.

We use an unconventional iteration counter. We increment the number of iterations if and only if an *unvisited vertex* is selected in that iteration. That is, the i^{th} iteration marks the iteration in which the i^{th} unvisited vertex is selected. For instance, the first iteration always discover an unvisited vertex, let this be v. In the following iteration, if v is selected again, it becomes a root and we do not count.

Else, an unvisited vertex u is selected and we increment the counter. Since a new fragment is created when an unvisited vertex is selected , the counter also counts the number of fragments.

Denote by A_i the *expected number* of available vertices at the start of the i^{th} iteration $(A_1 = n)$; by X_i the *expected size* of the fragment formed in that iteration. We have

$$X_i = 1 + (A_i - 1)p,$$

thus

$$A_{i+1} = A_i - X_i = A_i - \big\{1 + (A_i - 1)p\big\} = (A_i - 1)(1 - p).$$

If there remains one vertex, the number of fragments increments one last time. Suppose this occurs at the $(t+1)^{th}$ iteration i.e. $A_{t+1} = 1$. For convenience let $q = 1 - p$, we have

$$\begin{aligned} A_{t+1} &= (A_t - 1)q = \big\{(A_{t-1} - 1)q - 1\big\}q = (A_{t-1} - 1)q^2 - q \\ &= \big\{(A_{t-2} - 1)q - 1\big\}q^2 - q = (A_{t-2} - 1)q^3 - (q + q^2) = \dots \\ &= (A_1 - 1)q^t - \sum_{j=1}^{t-1} q^j = (n-1)q^t - \frac{q(1 - q^{t-1})}{1 - q} \\ &= (n-1)(1-p)^t - \frac{1-p}{p}(1 - (1-p)^{t-1}) = (n - 1 + \frac{1}{p})(1-p)^t - \frac{1}{p} + 1. \end{aligned}$$

To solve for $A_{t+1} = 1$, multiply both sides by p and simplify giving

$$(np - p + 1)(1 - p)^t = 1.$$

The left hand side is $(np - p + 1)(1 - p)^t \leq (np + 1)e^{-pt}$, with approximate equality provided that p is small i.e. $p = \omega/n = o(1)$ and using $e^{-x/(1-x)} \leq (1 - x) \leq e^{-x}$ $(0 < x < 1)$. Thus

$$(np + 1)e^{-pt} \sim 1 \quad \text{hence} \quad t \sim \frac{1}{p}\log(np + 1), \text{ the expected number of component.} \quad \square$$

Hypothetically, in the proximity graph the *degree* of a vertex might be an indication of its importance in the network. Intuitively a high degree vertex implies that it lies at a central position, in proximity of many other vertices. In the basic fragmentation process, vertices are selected with uniform distribution regardless of theirs properties. Thus, a low-degree vertex can absorb a higher-degree neighbour with positive probability, which might not be an ideal scenario. In which case, we can introduce new condition, for example, given a pair of vertices v, u

3. If $d(v) \geq d(u)$ v absorbs u; and vice versa.

Based on this idea and related ideas we are going to introduce and analyse some variations of the basic fragmentation algorithm in the following chapters.

Part II

Analysis of graph fragmentation algorithms

Chapter 3

Analysis of fragmentation algorithms on a cycle

3.1 Introduction

In this chapter, we make a theoretical analysis of the fragmentation algorithms introduced in Chapter 2 for the case of a cycle (or path). The analysis is made using a recurrence for the *expected number of fragments* formed from a cycle of length $L+1$ which eventually reduces to a path of length L.

The general procedure for the analysis of the fragmentation algorithms is as follows. For each variation of the algorithms, we describe the process on the input cycle or path. Where applicable, we give examples to demonstrate the difference between the variations. Next, we derive a recurrence for the expected number of fragments. The recurrence is then solved using a *generating function* (g.f). We then go through various steps including manipulating the g.f, solving differential equations, etc to analyse the g.f. Although most of the g.f(s) do not have an explicit series expansion based on a formula, we were able to extract the asymptotic growth of the coefficients of the g.f. The results of this study are summarised in Table 3.2, along with those obtained experimentally.

A supplementary introduction to the theory of *generating functions* is provided in Appendix A. The materials include g.f manipulations, analyticity and asymptotic of g.f and general procedure for solving differential equation, which are used extensively in the analysis.

3.2 General notation

Let $G(V,E)$ denote a graph G with V and E being the set of vertices and edges, respectively. Let $G'(V',E')$ be an empty graph. We then *copy* the vertex set of G to G' i.e. $V' \leftarrow V$, $E' \leftarrow \emptyset$. Objects in V are denoted *without a prime* symbol e.g. v. Objects in V' are denoted with a *prime* symbol e.g. v'. Denote by $Adj(v)$ the adjacency list of vertex v; and let $d(v)$ be its degree i.e. $d(v) = |Adj(v)|$.

Below we explain the procedure of the *basic fragmentation algorithm* (see Section 2.1.1). For each vertex $v \in V$, v is marked as *active*. We recurse

1. Select a pair of adjacent vertices
   (a) Select an *active* vertex v;
   (b) Select an *active* neighbour u, if v has no active neighbour $u = v$;
2. Let v absorbs u, meaning
   (a) Direct an edge (u, v);
   (b) u is marked as *inactive*;

In step one, a vertex v *is selected* uniformly at random (u.a.r) from the set of active vertex V. Next, v *chooses* an *active neighbour* u at random. If there is no active neighbour, v selects itself i.e. $u = v$. We regard v as the *main* vertex and u the *neighbour* vertex. Finally, with probability 1: v *absorbs* u; we denote this event in symbol by

$$(v \leftarrow u) \tag{3.1}$$

In step two, we make the following modifications to the graph G' and G. First, in G' we direct an edge between vertices v' and u' i.e. $e(u', v')$. There is a natural orientation of the new edge $e(u', v')$: u' is the *source* and v' be the *target*. This is to highlight the event (3.1). Next, G *is modified* as follows. Vertex u is *absorbed* and becomes *inactive*. The vertex u and its associated edges are *hidden* and unavailable for any future computation. In symbols

$$V = V \setminus \{u\}$$

and

$$\forall w \in Adj(u): \quad Adj(w) = Adj(w) \setminus \{u\}$$

If for any vertex v which has no *active neighbour*, then by (3.1) v absorbs itself $(v \leftarrow v)$ hence a *self-loop* is added $e(v', v')$. We regard v' as a *root*. The process is recursed until there remains no active vertex in G ($V = \emptyset$).

At the end of the algorithm, each vertex $v' \in G'$ has out-degree one. The output of the algorithm is a set of oriented unicyclic components consisting of trees, rooted on a directed cycle (possibly a self-loop).

3.2.1 Example

In Figure 3.1, we reproduce the example given in Section 2.1.1 using the notations introduced above.

3.3 Analysis of fragmentation algorithms on a cycle

In the following sections, we analyse the *basic graph fragmentation* introduced the previous section with the input as *cycle* graphs. In each section, we first describe a fragmentation algorithm then analyse them on a cycle. These sections are structured as follows:

Initialisation G and G':

Iteration 1: a absorbs b

Iteration 2: c absorbs d

Iteration 3: a becomes *root*

Iteration 4: c becomes *root*

Fig. 3.1 An example of a cycle with 4 vertices $V = (a,b,c,d)$ i.e. the left figure. Suppose the algorithm is applied according to the following steps:

1. a absorbs b; b is inactive. In symbol: $(a \leftarrow b)$.
2. c absorbs d; d is inactive. In symbol: $(c \leftarrow d)$.
3. a has no active neighbour; a *marks* itself and is inactive. $(a \leftarrow a)$
4. c has no active neighbour; c *marks* itself and is inactive. $(c \leftarrow c)$.

In each iteration the graph G and G' are presented in the left and right figure, respectively. The *dotted* line represent the edges of a vertex which becomes *inac tive* in that iteration. The result of the algorithm is the graph G' in the final iteration (bottom-left corner), in which there are two components.

1. A definition of the algorithm is given, the result is summarised in a *Proposition*.

2. The proof immediately follows the Proposition. Since the proof is quite lengthy, it is broken down into several smaller parts in the following order: *recurrence formulation*, *generating function formulation*, *solving differential equation* to obtain a closed form (if any) of the generating function and *analysis of the asymptotic growth of the coefficient of the generating function*.

3.3.1 Notation used in following analysis

The following notations are used for the rest of the chapter. As we only consider simple path, we suppress the symbol P that represents the path. We use an *unconventional path length*, that is let L denote the *length of a path* which is the *number of vertices* on the path P. Denote by N_L the *expected number of components* formed by applying the algorithm to a path of length L.

3.3.2 I.a: the basic fragmentation algorithm

We abbreviate the basic fragmentation algorithm described above as **I.a**. The pseudocode of this algorithm is given below. An example of this variation is provided in Figure 3.1.

Algorithm 2: Basic Fragmentation **I.a**

$\forall v \in V$, v is *active* **while** $V \neq \emptyset$ **do**
  Select an active vertex $v \in V$ u.a.r Select an active neighbour u if there is none $u = v$
  $(v \leftarrow u)$ u is inactive: $V = V \setminus \{u\}$ $\forall w \in Adj(u) : Adj(w) = Adj(w) \setminus \{u\}$
end

Proposition 1. *Suppose the input of the **I.a** algorithm is a cycle of length $L+1$. After the first iteration, the initial cycle breaks into a path of length L. Then, at the end of the algorithm, the expected number of components is*

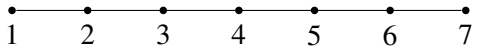
$$N_L \sim (L+1) \times (1 - e^{-1/2}) \approx (L+1) \times 0.39346$$

Let the input graph be a cycle of length $L+1$ i.e. it has $L+1$ vertices. In the first iteration, a random vertex is selected and absorbs one of its neighbour. Hence, exactly one vertex becomes *inactive* and the cycle is reduced to a *path* of length L.

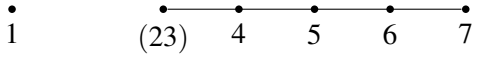
We reindex vertices on the path from $1, 2, \ldots, L$. In the *the second iteration*, assuming a vertex at index i, i.e. v_i is selected. Then for $1 < i < L$, v_i absorbs one of its neighbours i.e. v_{i-1} or v_{i+1} with equal probability $1/2$. Consequently, the path is broken into two *paths* of shorter length. Else if the selected vertex is at either end $i = 1$ or L, then v_i absorbs v_2 or v_{L-1} with probability 1, respectively.

As the process carries on, paths are gradually broken into smaller paths. Eventually broken paths are reduced to length 0, 1 or 2; for which the number of expected components can be calculated. Therefore, the expected number of components can be computed using a recurrence, which is derived in the section below.

Finally, we note a technical detail. Consider an example for a path of length $L = 6$



Suppose $i = 3$ (v_3 is selected). Then with probability $1/2$ the neighbour v_2 is chosen



Or, the neighbour v_3 is chosen

Important note. As illustrated above, the absorbed and absorbing vertices are treated as a single vertex in future calculations. This means that although we can estimate the expected number of components obtained from a path of length L, we are not able to obtain the distribution of component sizes using our methods.

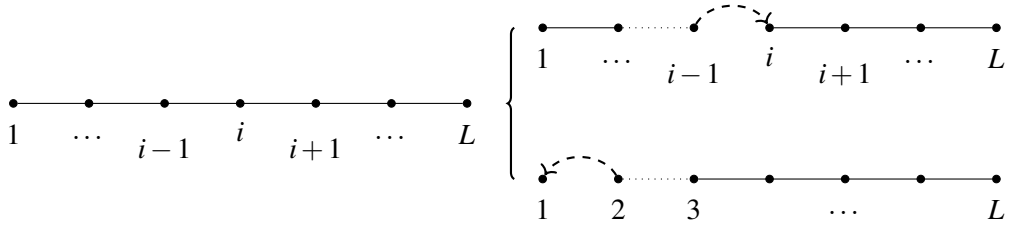


Fig. 3.2 An illustration of the recurrence. Assume the *input cycle is broken into a path* i.e. we are in the *second iteration*. Suppose the path has length L (left figure). In general, there are two cases could occur:

1. A vertex v_i $(2 \le i \le L-1)$ is selected with probability $(L-2)/L$. Suppose the event $(v_i \leftarrow v_{i-1})$ occurs (with probability $1/2$). The resulting structure consists of two paths of length $i-2$ and $L-i+1$, respectively. Note that if $i = 2$ and the event $(v_2 \leftarrow v_1)$ occurs, then the left component has length 0.
2. An end of the path i.e. $i = 1$ or L is selected with probability $2/L$. Suppose v_1 is selected, the event $Pr(v_1 \leftarrow v_2) = 1$. The result is two paths of length 1 and $L-2$, respectively.

3.3.2.1 Recurrence

Denote by N_L the *expected number of components* formed by applying the algorithm to a path of length L. Consider the recurrence for a general case illustrated in Figure 3.2. If the path has length 0 then $N_0 = 0$. If the path has length 1 then $N_1 = 1$; and $N_2 = 1$. Otherwise, if we start with a path of

length $L \geq 3$, then the following recurrence applies:

$$N_L = \sum_{i=1}^{L} N(i) \qquad (N_0 = 0, N_1 = 1, N_2 = 1) \tag{3.2}$$

where i is the index of vertices in the path. Note that if $j < s$ we assume $\sum_{i=s}^{j} N(i) = 0$. We have

$$N(i) = Pr(v = i)E(\text{number of components}|v = i)$$

hence

$$N(i) = \begin{cases} \frac{1}{L}(1 + N_{L-2}) & \text{if } i = 1, L \\ \frac{1}{L}\left(\frac{1}{2}[N_i + N_{L-i-1}] + \frac{1}{2}[N_{i-2} + N_{L-i+1}]\right) & \text{if } i = 2, 3..., L-1 \end{cases} \tag{3.3}$$

Expanding on N_L, we have

$$N_L = \sum_{i=1}^{L} N(i) = \frac{2}{L}(1 + N_{L-2}) + \sum_{i=2}^{L-1} N(i)$$

The sum on the right hand side can be expanded as

$$\begin{aligned} \sum_{i=2}^{L-1} N(i) = \frac{1}{2L}\Big(& N_0 + N_{L-1} + N_2 + N_{L-3} \\ & + N_1 + N_{L-2} + N_3 + N_{L-4} \\ & + \dots \\ & + N_{L-3} + N_2 + N_{L-1} + N_0\Big) \\ = \frac{1}{2L}(2\sum_{i=0}^{L-3} N_i + 2\sum_{j=2}^{L-1} N_j) &= \frac{1}{L}(\sum_{i=0}^{L-3} N_i + \sum_{j=2}^{L-1} N_j) \end{aligned}$$

Substituting this back into (3.2), we get

$$\begin{aligned} N_L &= \frac{2}{L}(1 + N_{L-2}) + \sum_{i=2}^{L-1} N(i) \\ &= \frac{1}{L}\Big(2 + 2N_{L-2} + \sum_{i=0}^{L-3} N_i + \sum_{j=2}^{L-1} N_j\Big) \\ &= \frac{1}{L}\Big(2 + N_0 + N_1 + N_{L-1} + N_{L-2} + 2\sum_{i=2}^{L-2} N_i\Big) \end{aligned} \tag{3.4}$$

It follows that

$$N_{L-1} = \frac{1}{L-1}\Big(2 + N_0 + N_1 + N_{L-2} + N_{L-3} + 2\sum_{i=2}^{L-3} N_i\Big) \tag{3.5}$$

Multiply (3.4) and (3.5) by L and $L-1$, respectively then subtract $(L-1)N_{L-1}$ from LN_L we get

$$LN_L-(L-1)N_{L-1}=N_{L-1}+2N_{L-2}-N_{L-3}$$
$$LN_L=LN_{L-1}+2N_{L-2}-N_{L-3}$$

Therefore, for $L\geq 3$

$$N_L=N_{L-1}+\frac{2}{L}N_{L-2}-\frac{1}{L}N_{L-3} \tag{3.6}$$

3.3.2.2 Generating function formulation

Next, we derive a generating function for N_L. Let $G(x)=\sum_0^\infty a_n x^n$ be an ordinary generating function (OGF) that generates N_L. Using the recurrence from (3.6), replacing N_L with a_n

$$a_n=a_{n-1}+\frac{2}{n}a_{n-2}-\frac{1}{n}a_{n-3} \qquad (n\geq 3; a_0=0, a_1=1, a_2=1) \tag{3.7}$$

Multiply the above equation by n and x^n and sum on n for $n\geq 3$, we get

$$\sum_3^\infty na_nx^n=\sum_3^\infty na_{n-1}x^n+2\sum_3^\infty a_{n-2}x^n-\sum_3^\infty a_{n-3}x^n \tag{3.8}$$

Three terms can be expressed in closed form involving G(x) as follows. Consider the single term in the left hand side from (3.8), we have the generating function $F(x)=\sum_0^\infty na_nx^n=xG'(x)$, where $G'(x)$ is the derivative of $G(x)$ (see Section A.1.1 for more details). Note that in $F(x)$ the sum is taken for $n\geq 0$. Hence, if the sequence starts from $n=3$ then $I(x)=\sum_{n=3}^\infty=F(x)-x-2x^2$. Thus, for all terms in (3.8) we have

- $$\sum_3^\infty na_nx^n=x.G'(x)-x-2x^2 \tag{3.9}$$
- $$2\sum_3^\infty a_{n-2}x^n=2.x^2G(x) \qquad \text{(right shifting, see Section A.1.1.2)} \tag{3.10}$$
- $$-\sum_3^\infty a_{n-3}x^n=-x^3G(x) \qquad \text{(right shifting)} \tag{3.11}$$

For the last term $\sum_{n\geq 3}na_{n-1}x^n$, we can manipulate the OGF using the *index multiply* and *right shift* to get

$$\sum_1^\infty na_{n-1}x^n=x\sum_0^\infty(n+1)a_nx^n=x\Big(\sum_0^\infty na_nx^n+\sum_0^\infty a_nx^n\Big)=x\Big(x.G'(x)+G(x)\Big)$$

Since the above series starts from $n=3$ as in (3.8), we need to subtract the term where $n=2$ which is $na_{n-1}x^n=2a_1x^2=2x^2$ because $a_1=1$ as stated in (3.7). Hence we have

$$\sum_3^\infty na_{n-1}x^n=x\Big(x.G'(x)+G(x)\Big)-2x^2$$

Substituting all these terms into (3.8) we have

$$\begin{aligned} x.G'(x) - x - 2x^2 &= x^2.G'(x) + x.G(x) - 2x^2 - 2x^2G(x) - x^3G(x) \\ x(1-x)G'(x) &= x(1+2x-x^2)G(x) + x \\ (1-x)G'(x) &= (1+2x-x^2)G(x) + 1 \end{aligned} \tag{3.12}$$

which is a first order linear differential equation. The procedure for solving such differential equations is (see Section A.4 for more details) to rewrite it in the form $y' + P(x).y = Q(x)$, i.e.

$$G'(x) + \frac{x^2-2x-1}{1-x}G(x) = \frac{1}{1-x} \tag{3.13}$$

Then the integrating factor is given by

$$\mu(x) = e^{\int P(x)dx} = e^{\int \frac{x^2-2x-1}{1-x}dx} = e^{2\log(x-1) - \frac{(x-1)^2}{2}}$$

Note that the function associated with $G(x)$ in (3.13) can be simplified to i.e. $\big((1-x)+2/(x-1)\big)$, which changes the second term in the exponential function to $x - x^2/2$. Nevertheless, this makes no difference and both forms give the same result in the end. The function y(x) is then given by

$$y(x) = \frac{\int \mu(x)Q(x)dx}{\mu(x)}$$

For the nominator we have

$$\int \mu(x)Q(x)dx = \int e^{(2ln(x-1) - \frac{(x-1)^2}{2})} \frac{1}{1-x}dx = \int (1-x)e^{(-\frac{(x-1)^2}{2})}dx = \int e^u du = e^{-\frac{(x-1)^2}{2}}$$

by substituting $-\frac{(x-1)^2}{2} = u$, $du = (1-x)dx$. Hence

$$G(x) = \frac{e^{(-\frac{(x-1)^2}{2})} + C}{e^{(2ln(x-1) - \frac{(x-1)^2}{2})}} = \frac{1}{(1-x)^2} + \frac{C.e^{\frac{(x-1)^2}{2}}}{(1-x)^2}$$

where C is the unknown constant of integration. We now solve for C using initial conditions of $G(x)$. Since $G(0) = a_0 = 0$, we have $0 = 1 + C.e^{-1/2}$, thus $C = -e^{-1/2}$. Finally, the generating function $G(x)$ is

$$G(x) = \frac{1}{(1-x)^2} - \frac{e^{x^2/2-x}}{(1-x)^2} \tag{3.14}$$

3.3.2.3 Expanding the OGF.

For the second term in (3.14), we can attempt to convolute the three series $e^{x^2/2}, e^{-x}$ and $\frac{1}{(1-x)^2}$ as follows (see convolution rule in Section A.1.1.4). The last OGF is well known, it is $\frac{1}{(1-x)^2} = \sum_0^\infty (n+1)x^n$. Therefore, we need to convolute $e^{\frac{x^2}{2}}$ and e^{-x}. Using the power series for the exponential

function (equation (A.5)), we get

$$e^{-x} = \sum_{0}^{\infty} \frac{(-1)^n . x^n}{n!} = 1 - x + \frac{x^2}{2!} - \frac{x^3}{3!} + \dots$$
$$e^{\frac{x^2}{2}} = \sum_{0}^{\infty} \frac{2^{(-n)} . (x^2)^n}{n!} = 1 + \frac{x^2}{2^1 . 1!} + \frac{x^4}{2^2 . 2!} + \frac{x^6}{2^3 . 3!} + \dots$$

The convolution of the two series e^{-x} and $e^{x^2/2}$ yields, let this be $E(x)$

$$E(x) = (1 - x + \frac{x^2}{2!} - \frac{x^3}{3!} + \dots)(1 + \frac{x^2}{2^1 . 1!} + \frac{x^4}{2^2 . 2!} + \frac{x^6}{2^3 . 3!} + \dots)$$

Notice the right hand side series contain only even exponent, the coefficients of such series are separated into odd and even case

$$[x^n]E(x) = \begin{cases} a_0 b_n + a_2 b_{n-2} + a_4 b_{n-4} + \dots + a_n b_0 & \text{(for n is } \textit{even}\text{)} \\ a_1 b_{n-1} + a_3 b_{n-3} + a_5 b_{n-5} + \dots + a_n b_0 & \text{(for n is } \textit{odd}\text{)} \end{cases}$$

where $[x^n]E(x)$ denote the n^{th} coefficient of the generating function $E(x)$. It is possible to write down a formula for the n^{th} coefficient for each even and odd case of this convoluted series. It would be quite problematic to obtain a further formula when it is convoluted with $1/(1-x)^2$. We expand this convoluted series above by a few terms

$$E(x) = 1 - x + x^2 - \frac{2x^3}{3} + \frac{5x^4}{12} - \frac{13x^5}{60} + \dots \tag{3.15}$$

Next, we convolute $E(x)$ and $1/(1-x)^2$. Let the series $E(x) = \sum_n c_n x^n$ and the series $F(x) = 1/(1-x)^2 = \sum_n d_n x^n$, the convolution of the two series has its coefficients given by

$$[x^n]E(x)F(x) = \sum c_k d_{n-k} = c_0 d_n + c_1 d_{n-1} + c_2 d_{n-2} + \dots + c_n d_0 \tag{3.16}$$
$$= \Big(c_0 d_n + c_2 d_{n-2} + \dots\Big) + \Big(c_1 d_{n-1} + c_3 d_{n-3} + \dots\Big) \tag{3.17}$$
$$= \Big(\sum_{i=0,2,4\dots}^{n} c_i (n+1-i)\Big) + \Big(\sum_{j=1,3,5\dots}^{n} c_j (n+1-j)\Big) \tag{3.18}$$

where c_i and c_j are the even and odd coefficient in (3.15). Note that the odd terms c_j are negative, thus the second summand is negative i.e. we are subtracting the second summand. We give an expansion of this triple-convoluted series (up to the fifth term, using (3.15) and (3.16)) below

$$E(x)F(x) = 1 + x + 2x^2 + \frac{7x^3}{3} + \frac{37x^4}{12} + \frac{217x^5}{60} + \dots$$

Finally, the OGF is now given by, using equation (3.14), subtracting the triple-convoluted series from the series $1/(1-x)^2$ i.e. $G(x) = 1/(1-x)^2 - E(x)F(x)$, for which we have the following

expansion

$$G(x) = x + x^2 + \frac{5}{3}x^3 + \frac{23}{12}x^4 + \frac{143}{60}x^5 + \dots \tag{3.19}$$

If we calculate the coefficients a_n for $3 \leq n \leq 5$ using equation (3.7) then the results are, recall that $a_0 = 1, a_1 = 1, a_2 = 1$,

$$\begin{aligned} a_3 &= a_2 + \frac{2}{3}a_1 - \frac{1}{3}a_0 = 1 + 2/3 = 5/3 \\ a_4 &= a_3 + \frac{2}{4}a_2 - \frac{1}{4}a_1 = 5/3 + 2/4 - 1/4 = 23/12 \\ a_5 &= a_4 + \frac{2}{5}a_3 - \frac{1}{5}a_2 = 23/12 + 2/3 - 1/5 = 143/60 \end{aligned}$$

which agree with (3.19).

3.3.2.4 Asymptotic growth of the coefficient of the generating function $G(x)$.

It seems problematic to obtain a closed form for $G(x)$ from equation (3.14). Alternatively, the coefficients of $G(x)$ can be estimated asymptotically using the analyticity of the function that is represented by the series i.e. as given in (3.14) . Section A.2.1 provide more details including the main theorems and an example regarding the analyticity and asymptotic of generating functions. Below we apply the methods to our problem.

The second function of G(x) can be rewritten as

$$f(x) = \frac{e^{-x+\frac{x^2}{2}}}{(1-x)^2} = \frac{e^{-1/2}}{(1-x)^2} + \frac{\left(e^{-x+\frac{x^2}{2}} - e^{-1/2}\right)}{(1-x)^2} \tag{3.20}$$

For the first function in (3.20), since $e^{-1/2}$ is a constant, $[x^n]f(x)$ is therefore $e^{-1/2}(n+1)$ where $[x^n]f(x)$ is the n^{th} coefficient of the series $f(x)$. The second term is an entire function of x, by Theorem 12 in Section A.2.1, its coefficients are $O(\varepsilon^n)$ for every positive ε, and therefore the coefficients of f(x) are

$$[x^n]f(x) = e^{-1/2}(n+1) + O(\varepsilon^n) \qquad (\forall \varepsilon > 0, n \to \infty)$$

The coefficients of G(x) are asymptotically

$$[x^n]G(x) = [x^n]\frac{1}{(x-1)^2} - [x^n]f(x) \sim (n+1)(1-e^{-1/2}) \approx (n+1) \times 0.39346 \quad \square$$

3.3.3 I.b: conditional fragmentation on degree of vertices

In the final section of previous chapter, we discuss that in the proximity graph the *degree* of a vertex might be an indication of its importance in the network. In *the basic fragmentation* that we analysed previously, vertices are selected with uniform distribution. Thus, a low-degree vertex can absorb a higher-degree neighbour with positive probability, which might not be ideal. To compensate for this,

we introduce a *conditional absorbing* operation. That is, given a pair of adjacent vertices v, u where v is the main vertex and u the neighbour then

$$\text{If } d(v) \geq d(u) \; v \text{ absorbs } u\text{; and vice versa.}$$

We abbreviate this variation **I.b**, the pseudocode is as follows

Algorithm 3: Basic Fragmentation Conditional on Vertices' Degree: **I.b**

$\forall v \in V$, v is active **while** $V \neq \emptyset$ **do**
  Select a vertex $v \in V$ u.a.r A vertex w (the vertex to be inactive) **if** *v has no active neighbour* **then**
    $(v \leftarrow v)$ $w = v$
  else
    Select an active neighbour u u.a.r **if** $d(v) \geq d(u)$ **then**
      $(v \leftarrow u)$ $w = u$
    else
      $(u \leftarrow v)$ $w = v$
    end
  end
  w is inactive: $V = V \setminus \{w\}$ $\forall y \in Adj(w) : Adj(y) = Adj(y) \setminus \{w\}$
end

The difference between **I.a** and **I.b** is that in the latter, the vertex with the higher degree always absorb the lower. On a cycle, this condition eliminates the case in which, previously, if either one of the two end-points of a path is selected the path would break it into two. Now, in this case, the path is only pruned i.e. its length reduced by 1. Hence, the previous recurrence is applicable with a minor tweak.

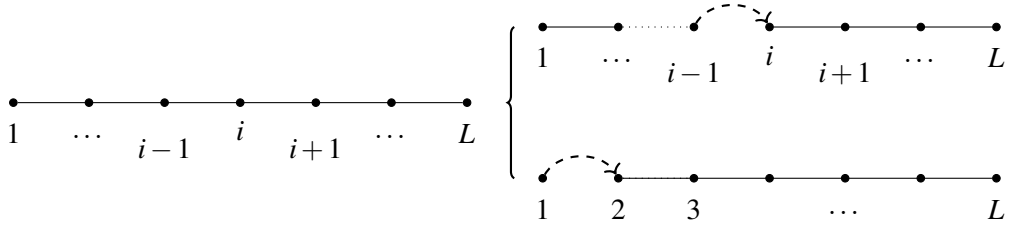


Fig. 3.3 An illustration of the recurrence. The difference between **I.a** (see Figure 3.2) and **I.b** (this version) is in the case where either end point is selected $i = 1$ or L, say $i = 1$ i.e. v_1 is selected. Then the event $(v_2 \leftarrow v_1)$ occurs with probability 1. This is because of the additional condition, $d(v_1) = 1 < d(v_2) = 2$ hence v_2 absorbs v_1. In which cases, v_1 goes *inactive* and the remaining path has length $L - 1$.

Proposition 2. *Suppose the input is a cycle of length $L+1$. After the first iteration, the remaining path has length L. Then, in the end the expected number of components is*

$$N_L \sim (L+1)\Big(1+e^{1/2}-\sqrt{2\pi}\,\mathrm{erfi}(1/\sqrt{2})\Big) \approx (L+1)\times 0.2588059$$

where erfi *is the imaginary error function.*

Let i denote the index of a vertex in the set $\{1,2\ldots,L\}$. Then similar to (3.2), we have $N_L = \sum_{i=1}^{L} N(i)$ (we assume that $\sum_{i=s}^{j} N(i) = 0$ for $j < s$). The function

$$N(i) = \begin{cases} \frac{1}{\mathbf{L}}\mathbf{N_{L-1}} & \text{if } i = 1, L \\ \frac{1}{L}\Big(\frac{1}{2}(N_i + N_{L-i-1}) + \frac{1}{2}(N_{i-2} + N_{L-i+1})\Big) & \text{if } i = 2,3...,L-1 \end{cases} \tag{3.21}$$

Thus summing on i gives

$$N_L = \frac{2}{L}N_{L-1} + \sum_{i=2}^{L-1} N(i) = \frac{1}{L}\Big(2.N_{L-1} + \sum_{i=0}^{L-3} N_i + \sum_{j=2}^{L-1} N_j\Big)$$

Similarly for N_{L-1}

$$N_{L-1} = \frac{1}{L-1}\Big(2.N_{L-2} + \sum_{i=0}^{L-4} N_i + \sum_{j=2}^{L-2} N_j\Big)$$

Multiply the equation for N_L and N_{L-1} with L and $L-1$, respectively. Then subtract $(L-1)N_{L-1}$ from LN_L we have

$$\begin{aligned} LN_L - (L-1)N_{L-1} &= 2N_{L-1} - 2N_{L-2} + N_{L-3} + N_{L-1} \\ N_L &= \frac{L+2}{L}N_{L-1} - \frac{2}{L}N_{L-2} + \frac{1}{L}N_{L-3} \end{aligned}$$

Replacing N_L with a_n, we have the following recurrence

$$a_n = \frac{n+2}{n}a_{n-1} - \frac{2}{n}a_{n-2} + \frac{1}{n}a_{n-3} \qquad (n \geq 3; a_0 = 0, a_1 = 1, a_2 = 1) \tag{3.22}$$

where the initial conditions a_0, a_1 and a_2 are calculated by hand. Multiply both side of the equation above by n and x^n, then sum on $n \geq 3$:

$$\sum_{3}^{\infty} n a_n x^n = \sum_{3}^{\infty}(n+2)a_{n-1}x^n - 2\sum_{3}^{\infty} a_{n-2}x^n + \sum_{3}^{\infty} a_{n-3}x^n \tag{3.23}$$

3.3.3.1 Generating function formulation

Let $G(x) = \sum_0^\infty a_n x^n$ be the OGF of a_n. Using the results in previous section particularly the series in (3.9), the summands in (3.23) can be written in terms of $G(x)$ as follows, note that as the index of n

starts from 3 we must subtract the terms where $n \leq 2$ as seen below

- $\sum_{3}^{\infty} n a_n x^n = xG'(x) - x - 2x^2$
- $\sum_{3}^{\infty}(n+2)a_{n-1}x^n = \sum_{3}^{\infty} n a_{n-1}x^n + 2\sum_{3}^{\infty} a_{n-1}x^n$
  $= x\Big(xG'(x) + G(x)\Big) - 2x^2 + 2\Big(xG(x) - x^2\Big)$
- $-2\sum_{3}^{\infty} a_{n-2}x^n = -2x^2G(x)$
- $\sum_{3}^{\infty} a_{n-3}x^n = x^3G(x)$

Substituting the OGFs into (3.23) yields

$$xG'(x) - x - 2x^2 = x^2G'(x) + 3xG(x) - 4x^2 - 2x^2G(x) + x^3G(x)$$
$$G'(x) + \frac{x^2 - 2x + 3}{x-1}G(x) = \frac{1-2x}{1-x}$$

which is a first order linear differential equation. Carrying out the usual procedure, the integrating factor is given by

$$\mu(x) = e^{\int P(x)dx} = e^{\int \frac{x^2-2x+3}{x-1}dx} = e^{(x-1)^2/2 + 2ln(x-1)} = (x-1)^2.e^{(x-1)^2/2}$$

Plug in the integrating factor, G(x) is given by

$$G(x) = y(x) = \frac{\int \mu(x)Q(x)dx}{\mu(x)} = \frac{\int (x-1)^2.e^{(x-1)^2/2}.\frac{(1-2x)}{1-x}dx}{(x-1)^2.e^{(x-1)^2/2}} \tag{3.24}$$

3.3.3.2 Finding the indefinite integral of an exponential function contains a polynomial

The integral in the nominator of $G(x)$ has an exponential function that contains a polynomial $e^{(x-1)^2/2}$ which is problematic to integrate. We also encounter this problem frequently in future sections. To tackle this, we introduce the following workaround which is also useful for future analysis. Let $F(x)$ be the function in the nominator of (3.24) of which we are trying to find the antiderivative

$$F(x) = (1-x)(1-2x)e^{(x-1)^2/2} = (2x^2 - 3x + 1)e^{(x-1)^2/2}$$

Let $e(x) = e^{(x-1)^2/2}$. Because $F(x)$ contain $e(x)$ and the latter has the form $e(x) = e^{f(x)}$ thus $e'(x) = f'(x)e^{f(x)}$ by the chain rule. Hence, the polynomial in $F(x)$ is potentially the product of differentiating the function in the exponential i.e. $f(x)$ up to some order. Thus, we can try to *guess* the antiderivative of $F(x)$ by differentiating $f(x)$ until the polynomial get to same order as the polynomial in $F(x)$ and

try to rewrite $F(x)$ in terms of the known derivatives. More specifically,

$$f'(x) = (x-1)e^{(x-1)^2/2}$$
$$f''(x) = (x^2-2x+2)e^{(x-1)^2/2}$$

Then, we rewritte $F(x)$ according to the above functions

$$F(x) = (2x^2-3x+1)e^{(x-1)^2/2} = [2(x^2-2x+2)+(x-1)-2]e^{(x-1)^2/2}$$

Hence, by linearity

$$\begin{aligned}\int F(x) &= 2\int (x^2-2x+2)+e(x)dx + \int (x-1)e(x)dx - 2\int e(x)dx \\ &= 2(x-1)e^{(x-1)^2/2} + e^{(x-1)^2/2} - 2\int e(x)dx \\ &= (2x-1)e^{(x-1)^2/2} - 2\int e(x)dx\end{aligned}$$

the remaining integral i.e. $\int e^{(x-1)^2/2}$ is a special function, which is discussed in the section below. Let $I(x) = \int e(x)dx$, using (3.24), the function $G(x)$ is

$$G(x) = \frac{2x-1}{(x-1)^2} - \frac{2.I(x)}{(x-1)^2 e(x)} + \frac{C}{(1-x)^2 e(x)}$$

where C is an unknown constant. To find C, we have $G(0) = 0$, $e(0) = e^{1/2}$ hence

$$G(0) = 0 = 1 - \frac{1}{e^{1/2}}\big(2I(0) - C\big) \quad \text{hence} \quad C = e^{1/2} + 2I(0)$$

Replace C into $G(x)$ gives

$$G(x) = \frac{2x-1}{(1-x)^2} - \frac{2I(x)}{(1-x)^2 e(x)} + \frac{1}{(1-x)^2 e(x)}(e^{1/2} + 2I(0)) \tag{3.25}$$

It seems problematic to obtain a formula for the coefficients of $G(x)$, hence alternatively we find the asymptotic growth of $[x^n]G(x)$.

3.3.3.3 The asymptotic of $[x^n]G(x)$

Denote the terms in equation (3.25) from left to right by $A(x), B(x), C(x)$. $A(x)$ is an OGF, it can be rewritten as $A(x) = \frac{2x}{(1-x)^2} - \frac{1}{(1-x)^2}$, and its coefficients are easily extracted $[x^n]A(x) = 2(n-1) - n = n-1$. The other functions are more problematic.

The function $C(x)$ has a pole at $x = 1$. Hence, according to Theorem 12.6 (Section A.2.1) the coefficients of $C(x)$ are well approximated by the principal part of its Laurent expansion [31] about the pole at $x = 1$. $C(x)$ contains two functions: $1/(x-1)^2$ and $1/e(x) = e^{-(x-1)^2/2}$; the former is

already an expansion about the pole. The latter is an entire function with a known expansion, hence

$$\begin{aligned}\frac{e^{-(x-1)^2/2}}{(x-1)^2} &= \frac{1}{(1-x)^2}\Big(1-\frac{(x-1)^2}{2}+\frac{(x-1)^4}{42!}-\frac{(x-1)^6}{83!}+\dots\Big)\\ &= \frac{1}{(1-x)^2}-\frac{1}{2}+\frac{1}{8}(x-1)^2-\frac{1}{48!}(x-1)^4+\dots\end{aligned} \tag{3.26}$$

The principal part of the expansion is the terms contain $(x-1)$ raised to negative powers i.e. $1/(1-x)^2$. We have that $[x^n]\frac{1}{(1-x)^2} = n+1$, thus $[x^n]C(x) \sim (n+1)[e^{1/2}+2I(0)]$.

The last function $B(x)$ is quite similar to $C(x)$, however with an extra function $I(x) = \int e^{(x-1)^2/2}$. $I(x)$ is a special function, called the *imaginary error function* erfi(z). It is an entire function which has the derivative $\frac{d}{dz}\text{erfi}(z) = \frac{2}{\sqrt{\pi}}e^{z^2}$, thus $I(x) = \sqrt{\frac{\pi}{2}}\text{erfi}(\frac{x-1}{\sqrt{2}})$. Hence we can expand it about $x=1$ using Taylor's theorem which yields

$$\sqrt{\frac{\pi}{2}}\text{erfi}(\frac{x-1}{\sqrt{2}}) = (x-1)+\frac{(x-1)^3}{6}+\frac{(x-1)^5}{40}+\dots \tag{3.27}$$

Finally, we convolute (3.26) with (3.27), but we only need to focus on the terms with negative powers of which there is only one single term: $\frac{1}{1-x}$. Thus, $[x^n]C(x) \sim [x^n]\frac{1}{x-1} = -1$.

Putting everything together, the asymptotic growth of the coefficients of $G(x)$ is

$$\begin{aligned}[x^n]G(x) &\sim (n-1)+2+(n+1)[e^{1/2}+2I(0)]\\ &= (n+1)\Big(1+e^{1/2}-\sqrt{2\pi}\,\text{erfi}(1/\sqrt{2})\Big) \approx (n+1)\times 0.2588059\end{aligned}$$

□

3.3.4 I.c: reverse of the absorb conditions of algorithm I.b

We modify the *absorb condition* of **I.b** as follows. Recall that the condition in I.b states that for a pair of selected adjacent vertices (v,u), the vertex with the *higher degree absorbs the lower*. In this version, we reverse this condition. More specifically, the vertex with *lower degree*, (regardless of whether it is the main or the selected neighbour), always absorbs the *higher degree* vertex. That is, suppose the pair of selected vertices is v_i, v_j

$$(v_i \leftarrow v_j) \quad \text{iff:} \quad d(v_i) < d(v_j)$$

Further, in the equality case $d(v_i) = d(v_j)$, the *main* vertex absorbs the chosen neighbour. We abbreviate this variation as **I.c**.

This modification changes the recurrence slightly by having an *additional case* when either the next-to-end-points v_2 or v_{L-1} is selected. Suppose it is v_2, then with probability $1/2$, v_2 selects v_1 the end vertex of the path; then the latter absorbs the former $(v_1 \leftarrow v_2)$ because $d(v_1) = 1 < d(v_2) = 2$, thus breaking original path into two paths of length 1 and $L-2$. Else v_2 chooses and absorbs v_3

$(v_2 \leftarrow v_3)$, this is because of the tie breaker rule; hence the path is divided into two paths of length 2 and $L-3$, respectively.

Proposition 3. *At the end of this algorithm, the expected number of components is*

$$N_L \sim (L+1)\Big(1-\frac{\sqrt{\pi}}{4}.\mathrm{erf}(1)-\frac{1}{2e}\Big) \approx (L+1)\times 0.44256$$

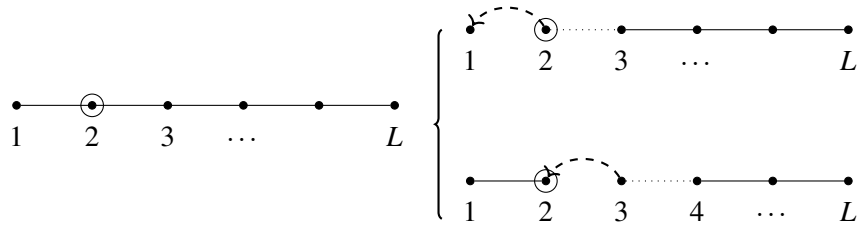


Fig. 3.4 An illustration of the scenario discussed previously. Suppose v_2 is selected. Then, v_2 selects v_1 with probability $1/2$. In which case, the event $(v_1 \leftarrow v_2)$ occurs with probability 1 because $d(v_1) = 1 < d(v_2) = 2$. Else, v_2 selects v_3 and $(v_2 \leftarrow v_3)$ by the tie break rule i.e. the main vertex absorbs the chosen neighbour.

The modification to the rules makes the following changes to the recurrence, which are highlighted in bold below

$$N(i) = \begin{cases} \frac{\mathbf{1}}{\mathbf{L}}(\mathbf{1}+\mathbf{N_{L-2}}) & \text{if } i = 1, L \\ \frac{\mathbf{1}}{\mathbf{L}}\Big(\frac{\mathbf{1}}{\mathbf{2}}(\mathbf{1}+\mathbf{N_{L-2}})+\frac{\mathbf{1}}{\mathbf{2}}(\mathbf{N_2}+\mathbf{N_{L-3}})\Big) & \text{if } \mathbf{i = 2, L-1} \\ \frac{1}{L}\Big(\frac{1}{2}(N_{i-2}+N_{L-i+1})+\frac{1}{2}(N_i+N_{L-i-1})\Big) & \text{if } i = 3,..,L-2 \end{cases}$$

Expanding on N_L we get

$$N_L = \frac{2}{L}(1+N_{L-2})+\frac{1}{L}(1+N_{L-2}+N_2+N_{L-3})+\sum_{i=3}^{L-2}N(i),$$

with the sum $\sum_{i=3}^{L-2}N(i) = (1/L)(\sum_{j=1}^{L-4}N(j)+\sum_{k=3}^{L-2}N(k))$. Continuing on as in previous section, by deriving N_{L-1} then subtracting N_{L-1} from N_L we arrive at the recurrence

$$N_L = \frac{L-1}{L}N_{L-1}+\frac{4}{L}N_{L-2}-\frac{2}{L}N_{L-3}$$

3.3.4.1 Generating function formulation

Substitue N_L with a_n, and let $G(x) = \sum a_n x^n$ be the OGF for a_n. We have the following recurrence:

$$a_n = \frac{n-1}{n}a_{n-1}+\frac{4}{n}a_{n-2}-\frac{2}{n}a_{n-3} \quad (n \geq 3, a_0 = 0, a_1 = 1, a_2 = 2) \tag{3.28}$$

Multiply by n and x^n and sum on $n \geq 3$ gives

$$\sum_3^\infty n a_n x^n = \sum_3^\infty (n-1) a_{n-1} x^n + 4 \sum_3^\infty a_{n-2} x^n - 2 \sum_3^\infty a_{n-3} x^n$$

We have encountered all these series in previous sections. Hence, using results from previous section (see Section 3.3.3.1), we arrive at the differential equation

$$G'(x) + \frac{2x(x-2)}{1-x} G(x) = \frac{1+x}{1-x} \tag{3.29}$$

Solving the differential equation, the integrating factor is: $\mu(x) = e^{\int 2x(x-2)/(1-x)dx} = e^{2ln(x-1)-(x-1)^2} = (x-1)^2 e^{-(x-1)^2}$. Then the nominator of $y(x) = \frac{\int \mu(x)Q(x)dx}{\mu(x)}$ is

$$\int (x-1)^2 e^{-(x-1)^2} . \frac{1+x}{1-x} dx = -\int (x^2-1) e^{-(x-1)^2} dx$$

which again involves finding the indefinite integral of $e^{-(x-1)^2}$. Thus, we apply the technique that was introduced in Section 3.3.3.2. First, we find the derivative of the exponential function $e(x) = e^{-(x-1)^2}$, up to second order, because the polynomial in the integral contains x^2,

$$\begin{aligned} e'(x) &= -2(x-1) e^{-(x-1)^2} \\ e''(x) &= (4x^2 - 8x + 2) e^{-(x-1)^2} \end{aligned}$$

The nominator of $y(x)$ can be tackled as follows

$$\begin{aligned} -\int (x^2-1) e^{-(x-1)^2} dx &= -\frac{1}{4} \int \Big((4x^2 - 8x + 2) + 8(x-1) + 2 \Big) e(x) dx \\ &= -\frac{1}{4} \Big(\int (4x^2 - 8x + 2) e(x) dx - 4 \int -2(x-1) e(x) dx + 2 \int e(x) dx \Big) \\ &= -\frac{1}{4} \Big(-2(x-1) e(x) - 4e(x) + 2 \int e(x) dx \Big) \\ &= \frac{1}{2} \Big((x+1) e(x) - \int e(x) dx \Big) \end{aligned}$$

Let $I(x) = \int e(x) dx$, we have the following form for $G(x)$

$$G(x) = \frac{\int \mu(x) Q(x) dx}{\mu(x)} = \frac{1}{2} \Big(\frac{x+1}{(x-1)^2} - \frac{I(x) + C}{(x-1)^2 e(x)} \Big) \tag{3.30}$$

where C is an unknown constant. To find C we have $G(0) = a_0 = 0$, further $e(0) = e^{-1}$, hence

$$0 = \frac{1}{2} \Big(1 - e\{I(0) + C\} \Big) \quad \text{hence } C = \frac{1}{e} - I(0)$$

Replace C into $G(x)$:

$$G(x) = \frac{1}{2}\Big(\frac{x+1}{(x-1)^2} - \frac{I(x)}{(x-1)^2 e(x)} + \frac{1/e - I(0)}{(x-1)^2 e(x)}\Big) \tag{3.31}$$

3.3.4.2 Asymptotic of $[x^n]$

The generating function that we derived in this section is quite similar to that of variant **I.b**. The only difference is the negative sign of the *special function*. Here $I(x) = \int e(x)dx = \int e^{-(x-1)^2}dx$, thus we get $I(x) = \frac{\sqrt{\pi}}{2}\operatorname{erf}(x-1)$, where $\operatorname{erf}(z)$ is the *error function*. The $\operatorname{erf}(z)$ is an entire function which has the derivative $\frac{d}{dz}\operatorname{erf}(z) = \frac{2}{\sqrt{\pi}}e^{-z^2}$. Hence, if we can rewrite $G(x)$ as follows

$$G(x) = \frac{1}{2}\frac{x+1}{(x-1)^2} - \frac{\sqrt{\pi}}{4}\frac{\big(\operatorname{erf}(x-1) - \operatorname{erf}(-1)\big)}{(x-1)^2 e(x)} - \frac{1}{2e}\frac{1}{(x-1)^2 e(x)}$$

To estimate the growth of G(x), again we apply the same method used in Section 3.3.3.3. The functions in $G(x)$ (except the first, which we can extract its coefficient) has a pole at $x = 1$. Thus, we expand the series at $x = 1$ to get its principal part

$$\begin{aligned}\frac{1}{(x-1)^2 e(x)} = \frac{e^{(x-1)^2}}{(x-1)^2} &= \frac{1}{(x-1)^2}\Big(1 + (x-1)^2 + \frac{(x-1)^4}{2!} + \frac{(x-1)^6}{3!} + \dots\Big) \\ &= \frac{1}{(x-1)^2} + 1 + \frac{(x-1)^2}{2!} + \frac{(x-1)^6}{3!} + \dots\end{aligned}$$

Thus, the last function in $G(x)$ has its coefficients given by $[x^n] \sim (1/2e)(n+1)$. For the function

$$A(x) = \frac{\sqrt{\pi}}{4}.\frac{\Big(\operatorname{erf}(x-1) - \operatorname{erf}(-1)\Big)}{(x-1)^2 e^{-(x-1)^2}}$$

the principal part of A(x) at $x = 1$ of is: $\frac{\sqrt{\pi}}{4}\frac{\operatorname{erf}(1)}{(x-1)^2} + \frac{1}{2(x-1)}$, hence $[x^n]A(x) \sim \frac{\sqrt{\pi}}{4}.\operatorname{erf}(1)(n+1) + \frac{1}{2}$.

Putting everything together, the asymptotic growth of the coefficients of $G(x)$ is

$$\begin{aligned}[x^n]G(x) &\sim \frac{1}{2}(2n+1) - \frac{\sqrt{\pi}}{4}.\operatorname{erf}(1)(n+1) + \frac{1}{2} - \frac{1}{2e}(n+1) \\ &= (n+1)\Big(1 - \frac{\sqrt{\pi}}{4}.\operatorname{erf}(1) - \frac{1}{2e}\Big) \approx (n+1) \times 0.44256\end{aligned}$$

□

3.4 Fragmentation by selecting the highest degree neighbour

In previous sections, we introduced some variations of the fragmentation algorithms where u.a.r sampling is used to select a pair of adjacent vertices in every iteration. In the following sections, we

study different variations of the fragmentation algorithms. In these variations, instead of selecting a neighbour at random, we select the neighbours which has the highest degree. This selection comes from the motivation to let the higher degree (which might be an indication of higher importance) vertex *absorbs* the lower degree vertex. Hence, these vertices could accumulate some advantage for future iterations.

Given a pair of adjacent vertices v and u where v is the *main* vertex and u is **the neighbour of v with highest degree**. With the motivation to let the higher degree vertex absorbs the lower, we introduce the following conditions, we abbreviate this variant as **II.c_(i)**,

1. If $d(v) < d(u)$ then u absorbs v i.e. $(u \leftarrow v)$;
2. If $d(u) > d(v)$ then v absorbs u i.e. $(v \leftarrow u)$;
3. Else select either u or v at random i.e. by flipping a coin.

In a cycle or a path, every vertex has degree 2 (except the two end-points in a path). Thus, it is likely the case that we have to *flip coins* to decide a *winner* (the absorber) for many iterations. Alternatively, we can remove the conditions and make one vertex wins by default. Hence we introduce the two following variations

- Algorithm **II.c_(ii)**: the *main* vertex wins by default i.e. $Pr(v \leftarrow u) = 1$.
- Algorithm **II.c_(iii)**: the *main* vertex loses by default i.e. $Pr(u \leftarrow v) = 1$.

Variation	Absorb rules Main: v, neighbour: u	Result
II.c_(i)	Conditional on $d(v)$ and $d(u)$	
	a) If $d(v) = d(u)$:	Pick at random
	b) If $d(v) > d(u)$:	*v absorbs u*
	c) If $d(v) < d(u)$:	*u absorbs v*
II.c_(ii)	v **wins**	*v absorbs u*
II.c_(iii)	u **wins**	*u absorbs v*

Table 3.1 Summary of the *absorbing rules* of the three variations of algorithm $II.c$ - (i), (ii) and (iii).

We summarise the results in this section in the proposition below.

Proposition 4. *Suppose the input is a cycle of length $L+1$. In the first iteration, it is reduced to a path of length L. Then, the expected number of components N_L, as a result of the algorithms, are:*

$$\begin{aligned} \textit{II.c\_(i):} \quad & N_L = (L+1) \times 0.2959 \\ \textit{II.c\_(ii):} \quad & N_L = (L+1) \times 0.2919 \\ \textit{II.c\_(iii):} \quad & N_L = (L+1) \times 0.3 \end{aligned}$$

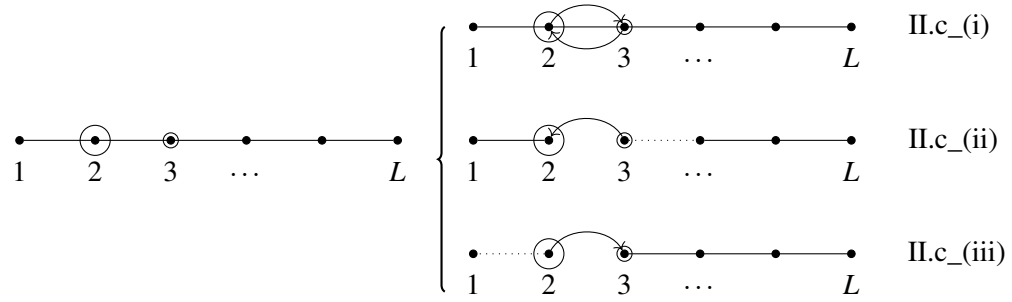


Fig. 3.5 The main difference between **II** versions compared to **I** variations is as follows. Suppose v_2 is selected (large circle); then **with probability** 1, vertex v_3 **is the chosen neighbour** (small circle). This is because v_2 have two neighbours v_1 and v_3 which has degree 1 and 2, respectively. Hence v_2 selects the latter, since we *always* select a vertex with highest degree. Whereas in previous variations (e.g. algorithm I.a), v_1 and v_3 is selected with equal probability $1/2$.

Furthermore, since $d(v_2) = d(v_3) = 2$, the *absorbed* vertex differs, depending on the *tie break rule*. More specifically, for each tie breaker rule, the result is

(i) Either v_2 and v_3 is selected at random. Thus $Pr(v_2 \leftarrow v_3) = Pr(v_3 \leftarrow v_2) = 1/2$.

(ii) The *main vertex* v_2 wins i.e. $Pr(v_2 \leftarrow v_3) = 1$.

(iii) The *neighbour vertex* v_3 wins i.e. $Pr(v_3 \leftarrow v_2) = 1$.

3.4.1 II.c_(i) - flip a coin if $d(v) = d(u)$

The notable difference in II.c_(i) is the occurrence of many tie-breakers as all vertices (except the two end-points) are of degree 2. For instances, consider a triplet $\{v_{i-1}, v_i, v_{i+1}\}$ ($2 \leq i-1$, $i+1 \leq L-1$), connected in pairs with $d(v_{i-1}) = d(v_i) = d(v_{i+1}) = 2$. Assuming the middle vertex v_i is selected, the following cases happen with equal probability:

$$\begin{cases} v_i \text{ picks } v_{i-1}\text{; then:} \begin{cases} (v_i \leftarrow v_{i-1}) & (Pr = 1/4) \\ (v_{i-1} \leftarrow v_i) & (Pr = 1/4) \end{cases} \\ v_i \text{ picks } v_{i+1}\text{; then:} \begin{cases} (v_i \leftarrow v_{i+1}) & (Pr = 1/4) \\ (v_{i+1} \leftarrow v_i) & (Pr = 1/4) \end{cases} \end{cases}$$

which yields the following recurrence.

3.4.1.1 Recurrence

Using notation from previous sections, let N_L be the expected number of components of a path with length $L \geq 0$. We can verify that $N_0 = 0$, $N_1 = 1$, $N_2 = 1$. Thus the base cases for the recurrence is

$$N_L = \sum_{i=1}^{L} N(i) \quad \Big(N_0 = 0, N_1 = 1, N_2 = 1\Big) \tag{3.32}$$

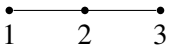
and

$$N(i) = \begin{cases} \frac{1}{L} N_{L-1} & \text{if } i = 1, L \\ \frac{1}{L}\Big[\frac{1}{2}(N_2 + N_{L-3}) + \frac{1}{2}(N_1 + N_{L-2})\Big] & \text{if } i = 2, L-1 \\ \frac{1}{L}\Big[\frac{1}{2}\big[\frac{1}{2}(N_{i-1} + N_{L-i}) + \frac{1}{2}(N_i + N_{L-i-1})\big] + \\ \quad \frac{1}{2}\big[\frac{1}{2}(N_{i-2} + N_{L-i+1}) + \frac{1}{2}(N_{L-i} + N_{i-1})\big]\Big] & \text{if } i = 3, \ldots L-2 \end{cases} \tag{3.33}$$

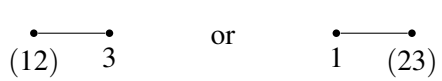
The third case looks rather messy, but can be rearranged to give

$$\frac{1}{L}\Big[\frac{1}{2}(N_{i-1} + N_{L-i}) + \frac{1}{4}(N_i + N_{L-i-1}) + \frac{1}{4}(N_{i-2} + N_{L-i+1})\Big] \tag{3.34}$$

Further, we extend the base cases to $L = 3, 4$ as the above recurrence does not take into account these cases. Consider a path with length $L = 3$:



Suppose vertex v_2 is selected i.e. $i = 2$, then using the second case in (3.33) yields $N_3 = 1/2$. However, the correct answer should be $N_3 = 1$ because if any vertex is selected, it would be absorbed by v_2 which has the highest degree.



The path is thus reduced to length $L = 2$, hence the expected number of components is $N_2 = 1$. The recurrence above fails in this case because both adjacent vertices of v_2 have degree 1 i.e. they are end points. This, however, is not included in the assumption of the recurrence. We also add to the base case $N_4 = N_3 + \frac{1}{2}N_1 = 3/2$, this is because later on the derivation of N_L is dependant on N_{L-1}.

We expand N_L as given in (3.32) and (3.33). Notice that when summing on i, the sums' variables are pair-wise interchangeable i.e. $\sum_3^{L-2} N(i) \equiv \sum_3^{L-2} N(L-i+1)$, etc. Putting everything together gives N_L

$$N_L = \frac{2}{L}N_{L-1} + \frac{1}{L}\big(N_1 + N_2 + N_{L-3} + N_{L-2}\big) + \frac{1}{L}\Big(\frac{1}{2}\sum_3^{L-2} N_{i-2} + \sum_3^{L-2} N_{i-1} + \frac{1}{2}\sum_3^{L-2} N_i\Big)$$

Multiply by L and redistribute the terms into the sums, we get

$$L.N_L = 2\sum_{i=1}^{L-1} N_i + \frac{1}{2}(-N_1 + N_2 + N_{L-3} - N_{L-2})$$

Similarly for N_{L-1}

$$(L-1)N_{L-1} = 2\sum_{j=1}^{L-2} N_j + \frac{1}{2}(-N_1 + N_2 + N_{L-4} - N_{L-3})$$

Subtracting $(L-1)N_{L-1}$ from LN_L yields

$$LN_L - (L-1)N_{L-1} = 2N_{L-1} - \frac{1}{2}(-N_{L-2} + 2N_{L-3} - N_{L-4}) \tag{3.35}$$

Finally, we get

$$N_L = \frac{L+1}{L}N_{L-1} + \frac{1}{2L}(-N_{L-2} + 2N_{L-3} - N_{L-4}) \tag{3.36}$$

3.4.1.2 Generating function formulation

Let $G(x) = \sum_0^\infty a_n x^n$ be an ordinary generating function for N_L. Using the recurrence from (3.36), replacing N_L with a_n yields

$$a_n = \frac{n+1}{n}a_{n-1} + \frac{1}{2n}(-a_{n-2} + 2a_{n-3} - a_{n-4}) \quad (n > 4, a_0 = 0, a_1 = 1, a_2 = 1, a_3 = 1, a_4 = 3/2)$$

Multiply both sides by n and x^n, then sum on $n \geq 5$ we get

$$\sum_5^\infty n a_n x^n = \sum_5^\infty (n+1)a_{n-1}x^n - \frac{1}{2}\sum_5^\infty a_{n-2}x^n + \sum_5^\infty a_{n-3}x^n - \frac{1}{2}\sum_5^\infty a_{n-4}x^n$$

Next, we rewrite all the terms in the equation above in terms of $G(x)$. We can use results from the previous sections (e.g. in Section 3.3.3.1), note that since the sum is taken for $n \geq 5$ we need to subtract the terms with smaller index. For instance,

- $\sum_5^\infty n a_n x^n = xG'(x) - (x + 2x^2 + 3x^3 + 6x^4)$
- $\sum_5^\infty n a_{n-1} x^n = x(xG'(x) + G(x)) - (2x^2 + 3x^3 + 4x^4)$
- $\sum_5^\infty a_{n-1} x^n = xG(x) - (x^2 + x^3 + x^4)$

and so on. Substituting all the sums according to $G(x)$

$$\begin{aligned} xG'(x) - (x + 2x^2 + 3x^3 + 6x^4) &= x(xG'(x) + G(x)) - (2x^2 + 3x^3 + 4x^4) + xG(x) \\ &\quad - (x^2 + x^3 + x^4) - \frac{1}{2}\big(x^2G(x) - (x^3 + x^4)\big) + x^3G(x) - x^4 \\ &\quad - \frac{1}{2}x^4G(x) \end{aligned}$$

After some algebra manipulations, the above simplifies to

$$G'(x) + G(x)\frac{x^3 - 2x^2 + x - 4}{2(1-x)} = \frac{x^3 - x^2 - 2x + 2}{2(1-x)}$$

To solve this first order differential, first calculate the integrating factor $\mu(x)$. Its exponent is

$$\int \frac{x^3 - 2x^2 + x - 4}{2(1-x)} = -\frac{x^3}{6} + \frac{x^2}{4} + 2\ln(x-1) \quad \text{or} \quad -\frac{(x-1)^2(2x+1)}{12} + 2\ln(x-1)$$

Thus, the integrating factor $\mu(x)$ is

$$\mu(x) = e^{-\frac{x^3}{6} + \frac{x^2}{4} + 2\ln(x-1)} = (x-1)^2 e^{-\frac{x^3}{6} + \frac{x^2}{4}}$$

Then $G(x)$ is given by $G(x) = \frac{\int \mu(x)Q(x)dx}{\mu(x)}$, of which the nominator of $G(x)$ is

$$\begin{aligned} \int \mu(x)Q(x)dx &= \int (x-1)^2 . e^{-\frac{x^3}{6} + \frac{x^2}{4}} \frac{x^3 - x^2 - 2x + 2}{2(1-x)} dx \\ &= -\frac{1}{2}\int (x^4 - 2x^3 - x^2 + 4x - 2)e^{-\frac{x^3}{6} + \frac{x^2}{4}} dx \end{aligned} \tag{3.37}$$

The integrand looks quite problematic for tackling directly. We apply the technique discussed in Section 3.3.3.2 to try to *guess* the function. As the integral contains an exponential function $e^{-x^3/6 + x^2/6}$ which contains a polynomial, we use this as our starting point. Next, we find the derivative of the exponential function up until a fourth-degree polynomial is obtained. From there we try to manipulate the integral according to the known derivatives. Particularly, let $e(x) = \exp\left(-\frac{x^3}{6} + \frac{x^2}{4}\right)$. The first and second order derivative of $e(x)$ are

$$\begin{aligned} e'(x) &= -\frac{x(x-1)}{2}e(x) \\ e''(x) &= \frac{(x^4 - 2x^3 + x^2 - 4x + 2)}{4}e(x) \end{aligned}$$

We can see that the second order derivative contains some terms which are quite similar to the terms in equation (3.37). Hence we can try to manipulate the integrand in (3.37) as follows (let's forget the

multiplying factor $-1/2$ for now):

$$\int (x^4-2x^3-x^2+4x-2)e(x)dx = \int \Big((x^4-2x^3+x^2-4x+2)-2(x^2-4x+2)\Big)e(x)dx$$

By linearity of integration, we get

$$\begin{aligned}\int (x^4-2x^3-x^2+4x-2)e(x)dx &= 4\int \frac{(x^4-2x^3+x^2-4x+2)}{4}e(x)dx - 2\int (x^2-4x+2)e(x)dx \\ &= 4e'(x)+4\int -\frac{x(x-1)}{2}e(x)dx+6\int xe(x)dx-4\int e(x)dx \\ &= 4e'(x)+4e(x)+6\int xe(x)dx-4\int e(x)dx\end{aligned}$$

Multiply back the factor $-1/2$, let $Y(x)=\int xe(x)dx$ and $I(x)=\int e(x)dx$, we have the final form the integral

$$\int (x^4-2x^3-x^2+4x-2)e(x)dx = -2e'(x)-2e(x)-3Y(x)+2I(x)+C$$

Substituting into $G(x)$, the generating function is then

$$G(x) = \frac{-2e'(x)-2e(x)-3Y(x)+2I(x)+C}{(x-1)^2e(x)} \tag{3.38}$$

where C is an unknown constant. To find C, we know $G(0)=0$, further $e'0)=0$, $e(0)=1$ thus

$$-2-3Y(0)+2I(0)+C=0, \quad \text{hence} \quad C=3Y(0)-2I(0)+2.$$

Finally

$$G(x) = \frac{-2e'(x)-2e(x)-3Y(x)+2I(x)+3Y(0)-2I(0)+2}{(x-1)^2e(x)} \tag{3.39}$$

As seen previously, the generating function for the sequence is too problematic to expand. Thus we evaluate the asymptotic of $[x^n]G(x)$. The function $G(x)$ has a simple pole at $x=1$. The polynomial in the denominator has the principal part at $x=1$ is $1/(x-1)^2$. All remaining functions *share* this factor. Further, all other functions involve exponential functions. Thus, to get the asymptotic of $G(x)$ we evaluate the other functions at $x=1$. This gives $e'(1)=0$, $e(1)=e^{1/12}$. Thus

$$\begin{aligned}[x^n]G(x) &\sim (n+1)(-2e^{1/12}-3Y(1)+2I(1)+3Y(0)-2I(0)+2)/e^{1/12} \\ &= (n+1)\Big[2(-e^{1/12}+1)-3\int_0^1 xe^{-x^3/6+x^2/4}+2\int_0^1 e^{-x^3/6+x^2/4}\Big]/e^{1/12} \\ &\approx (n+1)\Big(2(1-e^{1/12})-3\times 0.53018+2\times 1.04299)\Big)/e^{1/12} \\ &\approx (n+1)\times 0.2959\end{aligned}$$

□

3.4.2 II.c_(ii) - Tie breaker: definitive win for the main vertex

Let there be a pair of selected vertices *main* and *neighbour*. Recall that the main vertex is selected at random from the set of active vertices. The neighbour is the adjacent vertex of the main with the highest degree.

In this version, if $d(main) = d(\text{neighbour})$ we let the *main* vertex *wins by default*. Equivalently, the event that the main vertex absorbs the neighbour occur with probability 1. For example if v_2 is the main vertex (the next-to-end vertex), then v_3 is selected (because v_1 has degree 1) and the event $Pr(v_2 \leftarrow v_3) = 1$. This eliminates all random tie breakers as seen in the previous variant.

As usual, let N_L denotes the expected number of components of a path with length L ($N_0 = 0$, $N_1 = 1$ and $N_2 = 1$), thus

$$N_L = \sum_{i=1}^{L} N(i) \quad \Big(N_0 = 0, N_1 = 1, N_2 = 1\Big) \tag{3.40}$$

where i is the index of the selected vertex, hence

$$N(i) = \begin{cases} \frac{1}{L} N_{L-1} & \text{if } i = 1, L \\ \frac{1}{L}(N_2 + N_{L-3}) & \text{if } i = 2, L-1 \\ \frac{1}{L}\big(\frac{1}{2}(N_{i-2} + N_{L-i+1}) + \frac{1}{2}(N_i + N_{L-i-1})\big) & \text{if } i = 3, \ldots L-2 \end{cases} \tag{3.41}$$

This case looks much simpler compared to the previous recurrence. Checking the condition, we see that case where $L = 3$ satisfies the recurrence. Assuming if $L - 2 < 3$ the sum is empty, then

$$N_L = \frac{2}{L} N_{L-1} + \frac{2}{L}(N_2 + N_{L-3}) + \frac{1}{2L} \sum_{3}^{L-2} (N_{i-2} + N_{L-i+1} + N_i + N_{L-i-1}) \tag{3.42}$$

Which simplifies to

$$N_L = \frac{2}{L} N_{L-1} + \frac{2}{L}(N_2 + N_{L-3}) + \frac{1}{L} \sum_{3}^{L-2} [N_{i-2} + N_i]$$

Doing similarly for N_{L-1} then multiply N_L and N_{L-1} by L and $L-1$, respectively. Subtracting the latter from the former, we obtain

$$LN_L - (L-1).N_{L-1} = +2N_{L-1} - N_{L-2} + 2N_{L-3} - N_{L-4}$$

Hence

$$N_L = \frac{L+1}{L} N_{L-1} - \frac{1}{L} N_{L-2} + \frac{2}{L} N_{L-3} - \frac{1}{L} N_{L-4} \tag{3.43}$$

3.4.2.1 Generating function formulation

Let $G(x) = \sum_0^\infty a_n x^n$ be an ordinary generating function for N_L. Using the recurrence from (3.43) assuming $a_n = 0$ if $n < 0$, replacing N_L with a_n and multiplying both sides of (3.43) yields

$$na_n = (n+1)a_{n-1} - a_{n-2} + 2a_{n-3} - a_{n-4} \quad (n \geq 3, a_0 = 0, a_1 = 1, a_2 = 1) \tag{3.44}$$

Multiply by x^n and sum on $n \geq 3$

$$\sum_{n\geq 3} na_n x^n = \sum_{n\geq 3}(n+1)a_{n-1}x^n - \sum_{n\geq 3} a_{n-2}x^n + 2\sum_{n\geq 3} a_{n-3}x^n - \sum_{n\geq 3} a_{n-4}x^n$$

Rewritten the equation above in terms of $G(x)$ (using results from previous sections) we get

$$\begin{aligned} xG'(x) - (x+2x^2) &= x\big(xG'(x) + G(x)\big) - 2x^2 + (xG(x) - x^2) - x^2G(x) + 2x^3G(x) - x^4G(x) \\ x(1-x)G'(x) &= x(2 - x + 2x^2 - x^3)G(x) + x - x^2 \end{aligned}$$

Thus

$$G'(x) + \frac{x^3 - 2x^2 + x - 2}{1-x} G(x) = 1 \tag{3.45}$$

Following the usual routine, the integrating factor is

$$\mu(x) = e^{\int \frac{(x^3-2x^2+x-2)}{1-x}dx} = e^{2\ln(1-x) - x^3/3 + x^2/2} = (x-1)^2 e^{-x^3/3+x^2/2}$$

G(x) is then

$$G(x) = \frac{\int \mu(x)Q(x)dx}{\mu(x)} = \frac{\int (x-1)^2 e^{-x^3/3+x^2/2}dx}{(x-1)^2 e^{-x^3/3+x^2/2}} \tag{3.46}$$

Let $e(x) = e^{-x^3/3+x^2/2}$, we have the derivative: $e'(x) = (x-x^2)e^{-x^3/3+x^2/2}$ The nominator can then be manipulated as follows

$$\begin{aligned} \int (x-1)^2 e(x)dx &= -\int (x - x^2 + x - 1)e(x)dx \\ &= -e(x) - \int xe(x)dx + \int e(x)dx + C \end{aligned}$$

Let $I(x) = \int xe(x)dx$ and $Y(x) = \int e(x)dx$. We have that at $x = 0$, $G(0) = 0$, $e(0) = 1$, hence $-1 - I(0) + Y(0) + C = 0$ which implies $C = I(0) - Y(0) + 1$. Therefore

$$G(x) = \frac{-e(x) - I(x) + Y(x) + I(0) - Y(0) + 1}{(x-1)^2 e(x)}$$

For the asymptotic of the coefficients of $G(x)$, because $e(x)$, $I(x)$ and $Y(x)$ are entire functions, using the technique used in Section A.2.1.2, we calculate the value $e(1), I(1), Y(1)$ and hence

$$[x^n]G(x) \sim (n+1) \times \Big(-e^{1/6} - \int_0^1 xe(x)dx + \int_0^1 e(x)dx + 1 \Big) / e^{1/6} \approx (n+1) \times 0.2919$$

□

3.4.3 II.c_(iii) - Tie breaker: default loss for the main vertex

This is a *reversed* case of the previous type. Suppose we selected a pair of adjacent vertices: *main* and *neighbour*. Then the event: the **neighbour** *absorbs* the **main** occurs with probability 1.

For instance, suppose that v_2 is selected i.e. it is the main vertex. Next, v_2 selects a neighbour which has the highest degree: v_3. It follows that the event $(v_3 \leftarrow v_2)$ occurs with probability 1. This is because the absorbing condition states that the *selected* vertex is always *absorbed* by the neighbour. Thus, in each iteration, the *selected* vertex always become *inactive*. Figure 3.6 provides an illustration of all possible cases in this algorithm.

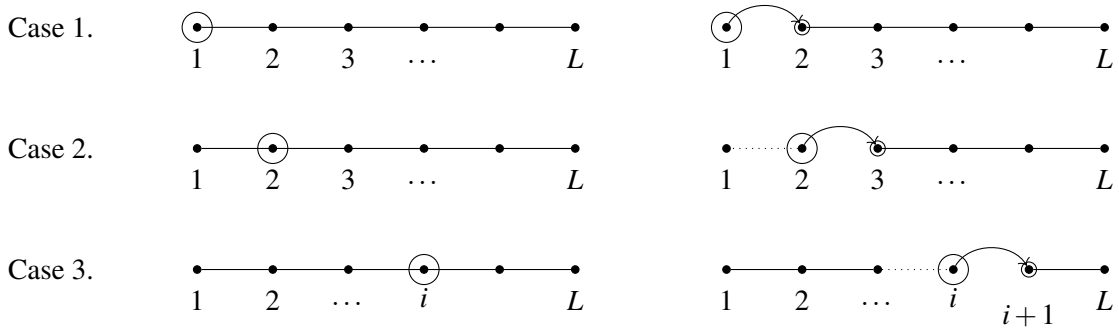


Fig. 3.6 For this variation, if the main vertex is

1. v_1 then $(v_2 \leftarrow v_1)$ occurs with probability 1;
2. v_2 then $(v_3 \leftarrow v_2)$ occurs with probability 1;
3. v_i for $3 \leq i \leq L-2$ then the selected neighbour is v_{i-1} or v_{i+1} with probability 1/2. In either case, the *selected* vertex is *always absorbed*. Hence, the original path always reduce to two shorter paths of length $i-1$ and $L-i$.

3.4.3.1 The Recurrence

As usual, let N_L denotes the expected number of components of a path with length L then it can be verified that $N_0 = 0, N_1 = 1, N_2 = 1$. Furthermore, we need to add these cases $N_3 = 1$ and $N_4 = 3/2$. This is because $L = 3$ is a special case i.e. there is a single vertex with degree two, but we assume the

equations are symmetric (there exists both vertices 2 and $L-2$) when expanding N_L below.

$$N_L = \sum_{i=1}^{L} N(i) \quad \Big(N_0 = 0, N_1 = 1, N_2 = 1\Big) \tag{3.47}$$

where i is the index of the selected vertex, furthermore

$$N(i) = \begin{cases} \frac{1}{L} N_{L-1} & \text{if } i = 1, L \\ \frac{1}{L}(N_1 + N_{L-2}) & \text{if } i = 2, L-1 \\ \frac{1}{L}\big(N_{i-1} + N_{L-i}\big) & \text{if } i = 3, \ldots L-2 \end{cases} \tag{3.48}$$

where the third case has been simplified, its original form is, for $3 \leq i \leq L-2$,

$$\frac{1}{L}\Big(\frac{1}{2}\big(N_{i-1} + N_{L-i}\big) + \frac{1}{2}\big(N_{i-1} + N_{L-i}\big)\Big) = \frac{1}{L}\big(N_{i-1} + N_{L-i}\big)$$

The recurrence for this case looks much simpler compares to some other variants. We expand N_L as usual

$$N_L = \frac{2}{L}(N_{L-1} + N_1 + N_{L-2}) + \frac{1}{L}(\sum_{3}^{L-2} N_{i-1} + \sum_{3}^{L-2} N_{L-i}) = \frac{2}{L}\sum_{1}^{L-1} N_i$$

Multiply by L and subtract $(L-1)N_{L-1}$ from LN_L yields

$$N_L = \frac{L+1}{L} N_{L-1}$$

Similarly, we have $N_{L-1} = \frac{L}{L-1} N_{L-2}$. Thus, telescoping

$$N_L = \frac{L+1}{L}\frac{L}{L-1}\cdots\frac{7}{6}\frac{6}{5}N_4$$

$N_4 = 3/2$, hence

$$N_L = (L+1)\frac{3}{10}$$

□

3.4.4 II.h - highest degree vertex absorbs lowest degree neighbour

On a cycle, this algorithm proceeds as follows. With probability 1, the input cycle of length $L+1$ is reduced to a path of length L. In the following iterations, the two end vertices are not selected, unless the path has length $L \leq 2$. For $L \geq 3$, an *inner-vertex* v_i $2 \leq i \leq L-1$ is chosen at random. If the main vertex is one of the *next-to-end* points i.e. $i = 2, L-1$, it absorbs the end vertex i.e. $i = 1, L$ with probability 1. Thus the path is reduced to length $L-1$. In other cases, the path breaks into two smaller paths, and the following recurrence applies.

Algorithm 4: Fragmentaion **II.h**

$\forall v \in V$, v is *active*
while $V \neq \emptyset$ **do**
  Select the *highest degree* vertex $v \in V$ (if tie pick u.a.r) A vertex w (to be inactive) **if** *v has no active neighbour* **then**
    $(v \leftarrow v)$ $w = v$
  else
    select the *lowest degree* neighbour u $(v \leftarrow u)$ $w = u$
  end
  w is *inactive*: $V = V \setminus \{w\}$ $\forall y \in Adj(w) : Adj(y) = Adj(y) \setminus \{w\}$
end

3.4.4.1 The Recurrence

Let N_L denotes the expected number of components of a path with length L, assuming for $j < s$ the sum $\sum_{i=s}^{j} N(i) = 0$. We have

$$N_L = \sum_{i=1}^{L} N(i) \quad \Big(N_0 = 0, N_1 = 1, N_2 = 1\Big) \tag{3.49}$$

where i is the index of the main vertex, and

$$N(i) = \begin{cases} \frac{1}{L-2} N_{L-1} & \text{if } i = 2, L-1 \\ \frac{1}{L-2}\Big(\frac{1}{2}\big(N_{i-1} + N_{L-i-1}\big) + \frac{1}{2}\big(N_{i-2} + N_{L-i+1}\big)\Big) & \text{if } i = 3, \ldots, L-2 \end{cases} \tag{3.50}$$

Expanding on N_L gives

$$(L-2)N_L = 2N_{L-1} + \sum_{3}^{L-2} N_i + \sum_{3}^{L-2} N_{i-2}$$

Carry out similarly, we obtain an equation for N_{L-1}

$$(L-3)N_{L-1} = 2N_{L-2} + \sum_{3}^{L-3} N_i + \sum_{3}^{L-3} N_{i-2}$$

Subtract $(L-3)N_{L-1}$ from $(L-2)N_L$ then rearrang the terms, we have the recurrence for N_L

$$N_L = \frac{L-1}{L-2} N_{L-1} + \frac{1}{L-2} N_{L-4} - \frac{1}{L-2} N_{L-2} \tag{3.51}$$

3.4.4.2 Generating function formulation

Let $G(x)$ be the OGF that generates N_L, assuming $a_i = 0$ for $i < 0$ we have

$$a_n = \frac{n-1}{n-2} a_{n-1} - \frac{1}{n-2} a_{n-2} + \frac{1}{n-2} a_{n-4} \quad (n \geq 3, a_0 = 0, a_1 = 1, a_2 = 1)$$

Multiply both side by $(n-2)$ and x^n then sum on $n \geq 3$ yields

$$\sum_{n\geq 3}(n-2)a_n x^n = \sum_{n\geq 3}(n-1)a_{n-1}x^n - \sum_{n\geq 3}a_{n-2}x^n + \sum_{n\geq 3}a_{n-4}x^n \tag{3.52}$$

Using results from previous sections, we rewrite the terms in the equation above according to $G(x)$

$$\begin{aligned} xG'(x) - (x+2x^2) - 2[G(x) - (x+x^2)] &= x(xG'(x)+G(x)) - 2x^2 - (xG(x)-x^2) + \\ &\quad x^4G(x) - x^2G(x) \\ G'(x) + \frac{x^4-x^2+2}{x(x-1)}G(x) &= \frac{x+1}{x-1} \end{aligned}$$

Following the routine for solving the first order differential equation, the integrating factor is given by

$$\begin{aligned} \mu(x) &= \exp\Big(\int \frac{x^4-x^2+2}{x(x-1)}dx\Big) = \exp\Big(\int \frac{x^2(x-1)(x+1)+2}{x(x-1)}dx\Big) \\ &= \exp\Big(\int x^2+x-\frac{2}{x}+\frac{2}{x-1}dx\Big) = \exp\big(x^3/3+x^2/2+2\ln(1-x)-2\ln(x)\big) \\ &= \frac{(x-1)^2}{x^2}e^{x^3/3+x^2/2} \end{aligned}$$

Let $e(x) = e^{x^3/3+x^2/2}$, we have

$$G(x) = \frac{\int \mu(x)Q(x)dx}{\mu(x)} = \frac{\int \frac{(x-1)^2}{x^2}\frac{(x+1)}{(x-1)}e(x)dx + C}{(x-1)^2e(x)/x^2} = \frac{\int (x^2-1)e(x)/x^2dx + C}{(x-1)^2e(x)/x^2}$$

If we simplify this, we will have two integrands: $\int e(x)dx$ and $\int e(x)/x^2dx$. The latter is undefined at 0, which is problematic later on when solving the initial conditions to find the constant C by substituting $x=0$. Thus alternatively, notice the derivative $\big(\frac{e(x)}{x}\big)' = (x^3+x^2-1)e(x)/x^2$. Then we can rewrite $G(x)$ to take advantage of this

$$\begin{aligned} G(x) &= \Big(\int (x^3+x^2-1)e(x)/x^2dx - \int xe(x)dx\Big)\frac{x^2}{(x-1)^2e(x)} \\ &= \Big(e(x)/x - \int xe(x)dx + C\Big)\frac{x^2}{(x-1)^2e(x)} \end{aligned}$$

And we have removed the pole at 0, which is useful for finding C since we are going to substitute $x=0$. Now $G(0)=0$ with any value of C, we need to use other initial conditions, $G'(0)=a_1=1$ or $G''(0)=2a_2=2$. To proceed with the derivatives, let $I(x)=\int xe(x)dx$

$$G(x) = \underbrace{\frac{x}{(x-1)^2}}_{\alpha(x)} - I(x)\underbrace{\frac{x^2}{(x-1)^2e(x)}}_{\gamma(x)} + C\underbrace{\frac{x^2}{(x-1)^2e(x)}}_{\gamma(x)}$$

Thus the first derivative of $G(x)$ is given by

$$\begin{aligned} G'(x) &= \alpha'(x) - [I'(x).\gamma(x) + I(x).\gamma'(x)] + C.\gamma'(x) \\ &= -\underbrace{\frac{x+1}{(x-1)^3} - \frac{x^3}{(x-1)^2}}_{\beta(x)} - I(x).\underbrace{\frac{x(x^4-x^2-2)}{(x-1)^3}}_{\gamma'(x)} + C.\underbrace{\frac{x(x^4-x^2-2)}{(x-1)^3}}_{\gamma'(x)} \end{aligned}$$

It can be seen that $G'(0) = 1$ which satisfies the initial condition, however does not provide a hint on C. Thus, unfortunately we need to proceed to the second order derivative. Let

$$G''(x) = \underbrace{\beta'(x)}_{A} - \underbrace{I'(x).\gamma'(x)}_{B} - \underbrace{I(x).\gamma''(x)}_{D} + \underbrace{C.\gamma''(x)}_{E}$$

After some tedious work, we have that

$$\begin{aligned} \beta'(x) &= -\frac{x^4 - 4x^3 + 3x^2 - 2x - 4}{(x-1)^4} \\ \gamma''(x) &= -\frac{x^8 - 2x^6 - 2x^5 + 8x^4 - 5x^2 + 4x + 2}{(x-1)^4 e(x)} \end{aligned}$$

With $x = 0$, we get $e(0) = 1$, $\beta'(0) = 4$, $\gamma'(0) = 0$ and $\gamma''(0) = 2$. It follows that $A = 4$, $B = 0$, $D = 2.I(0)$ and finally $E = 2C$. Thus, substitute all these values in to $G''(x)$ we get

$$G''(x) = 4 - 2I(0) + 2C = 2, \quad \text{hence} \quad C = I(0) - 1$$

Finally, we have

$$G(x) = \big(e(x)/x - I(x) + I(0) - 1\big)\frac{x^2}{(x-1)^2 e(x)} \tag{3.53}$$

With $x = 1$ is the only simple pole for $G(x)$, to get the asymptotic of the coefficient

$$[x^n]G(x) = (n+1)\big(e^{5/6} - \int_0^1 xe(x)dx - 1\big)/e^{5/6} \approx (n+1)(2.3 - 0.7558 - 1)/2.3 = (n+1) \times 0.2366$$

□

3.5 Conclusion

In this chapter, we studied some variations of graph fragmentation algorithms on cycle graphs. Using recurrences and generating function, we computed the expected number of components for each variation. It is seen that altering the selection rules gives different generating functions and hence the expected number of components varies.

We summarise the results in this chapter and also Chapter 4 in Table 3.2 below. In which, the *Expected* column shows the asymptotic proportion of rooted uni-cyclic components for a cycle C_n of

length n i.e.

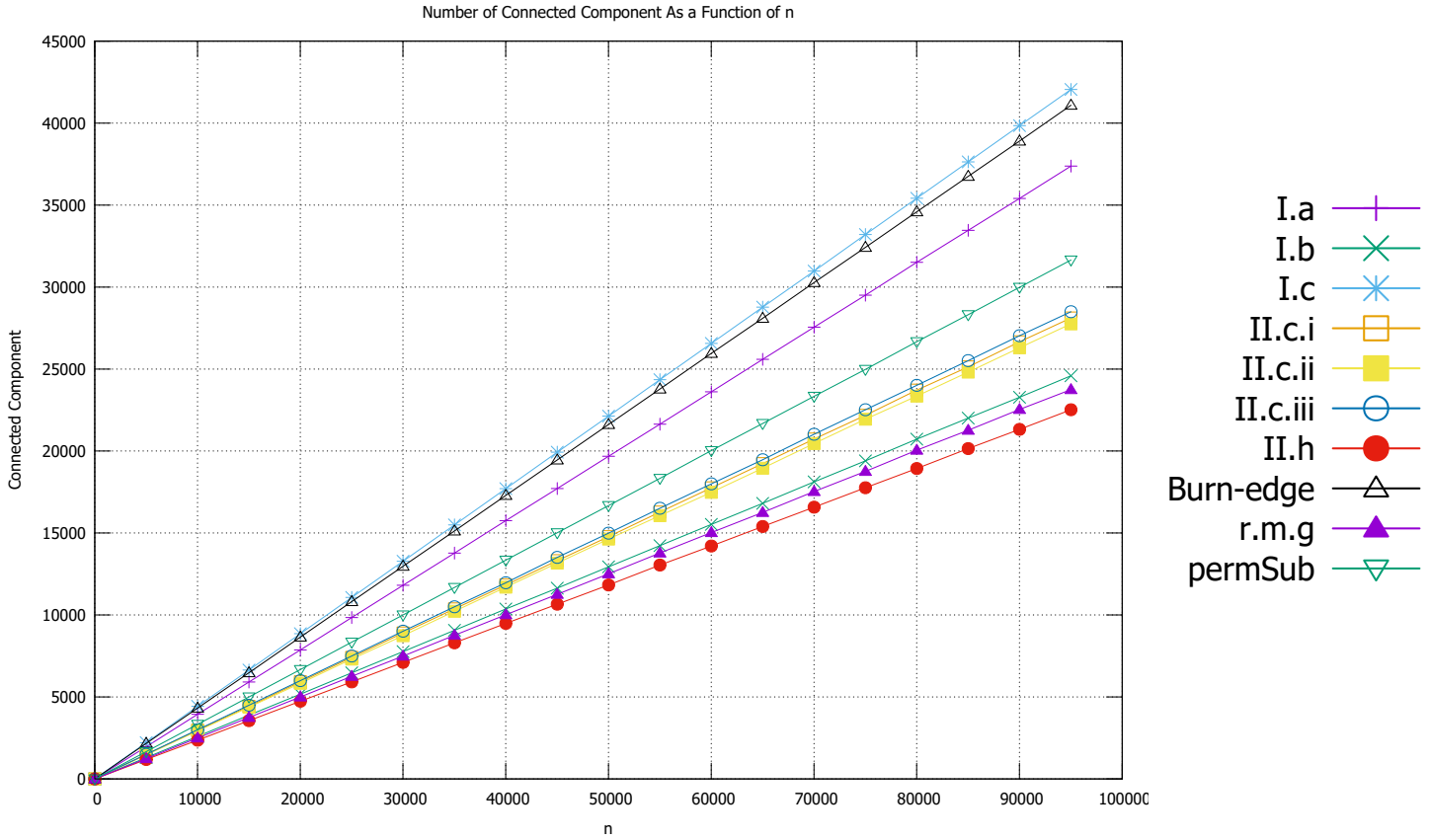
$$\text{Expected Coefficient} = \lim_{n\to\infty} \frac{E[\text{number of components of } C_n]}{n}$$

Additionally, we conduct experiments by applying the algorithms to cycles with the number of vertices scales from $n = 5000, \ldots, 100000$ vertices. For each algorithm, for each value of n we repeat the experiment for 10 times, then record the averaged number of components.

The experimental numbers of components obtained are plotted in Figure 3.7. Using which, we calculate the *experimental proportio*n of rooted uni-cyclic components over n and show this under the *Observed* column.

Fragmentation Variation	Expected	Observed
I.a	0.39346	0.3935
I.b	0.2588	0.2588
I.c	0.4425	0.4426
II.c_(i)	0.2959	0.2959
II.c_(ii)	0.2919	0.2921
II.c_(iii)	0.3	0.2999
II.h	0.2366	0.2370
Burn-edge*	0.4323	0.4326
Permutation subgraph*	1/3	0.3331
$G_{\text{1-out}}$*	1/4	0.2499

Table 3.2 Table of Summary: the expected and observed value of the number of components.
*The models: Burn-edge, *Permutation subgraph* and $G_{\text{1-out}}$ are introduced in Chapter 4.



(a) (b) Legends

Fig. 3.7 Experiments of fragmentations and mappings on a cycle. For each value of n, we *execute each algorithm for* 10 *times* and record the number of components. The lines show the *averaged number of components* as a function of n i.e. the number of vertices in the input cycle. Each line corresponds to a different algorithm i.e. see legends in the right figure. The testing algorithms also include some fragmentation algorithms introduced in future sections.

We conclude that with *a minor modification* to the general procedure, we can vary the number of components *substantially*.

Chapter 4

Permutation subgraphs

In the previous chapter, we studied some variations of fragmentation algorithms. These variations share a similar iterative procedure: select a vertex (the *main* vertex), then one of its neighbours (if any) and *remove* one vertex among the pair from the current graph. A common problem is that the introduction of various selection schemes e.g. degree comparison, etc. complicates the computation. Also, further complications come from letting the main vertex absorb the neighbour, since future selections are dependant on the outcome of previous iterations.

We notice that in variation II.c_(iii) (see Section 3.4.3), if we let the main vertex *point to* a neighbour and then go *inactive*, the recurrence becomes easier to derive. Motivated by this, we introduce the following process.

4.1 Permutation subgraph process

Consider the algorithm as given by Algorithm 5 below. Suppose the input is a simple, undirected graph with $G(V,E)$, we observe that

Proposition 5. *Each main vertex is sampled uniformly without replacement from the vertex set. Thus, the order of selection of main vertices is a permutation of* V.

We walk through the procedure of the algorithm. We start by selecting a vertex v u.a.r. Then v selects a random neighbour u and $(u \leftarrow v)$. Else, if $d(v) = 0$ then v becomes a root i.e. $(v \leftarrow v)$. In either case, v becomes *inactive*, *regardless of* u. Hence, we see that in every iteration a vertex is selected and removed from the vertex set. In other words, the *main* vertices are *sampled uniformly without replacement* from the vertex set V. Equivalently, the order in which the main vertices are selected is a permutation of V.

Algorithm 5 provides the pseudocode for the above process.

Algorithm 5: Permutation subgraph (permSub)

while $V \neq \emptyset$ **do**
  Select an active vertex $v \in V$ u.a.r ;
  if *v has no active neighbour* **then**
    $(v \leftarrow v)$;
  else
    Select an active neighbour u u.a.r;
    $(u \leftarrow v)$;
  end
  v is *inactive*: $V = V \setminus \{v\}$;
  $\forall u \in Adj(v) : Adj(u) = Adj(u) \setminus \{v\}$;
end

4.1.1 Definition

Let $G = G(V,E)$ be a simple, undirected graph with vertex set V. Let π be a random permutation of V i.e. $\pi = \big(\sigma(1), \sigma(2), \ldots, \sigma(n)\big)$ where each $\sigma(i)$ yields a unique vertex $v \in V$. We denote vertices in π as v_i where i is its index in π. A permutation subgraph H is constructed as follows. For each vertex v_i, we direct a *single forward edge* from v_i to one random vertex v_j i.e. (v_i, v_j), if and only if $i < j$ and there exists an edge $(v_i, v_j) \in E(G)$. In other words, we successively inspect each vertex v_i in π. For each v_i, we connect it to one of its neighbour v_j chosen at random if and only if v_j *succeeds* v_i in the permutation i.e. $i < j$. If, for any vertex v_i, there is no edge between v_i and every vertex that succeeds it in π, then v_i attaches to itself, and regarded as a root vertex.

Fig. 4.1 An example of algorithm, the input graph is shown on the left with $V = \{a,b,c,d\}$. Suppose a permutation is generated $\pi = (a,c,b,d)$ hence we have $v_1 = a, v_2 = c, v_3 = b, v_4 = d$. Then the following edges are connected, in order:

1. $e(a,b)$ i.e. $e(v_1, v_3)$;
2. $e(c,b)$ i.e. $e(v_2, v_3)$;
3. $e(b,b)$ hence b or v_3 is a root;
4. $e(d,d)$ hence d or v_4 is a root.

The result permutation subgraph H is shown on the right.

4.1.2 Comparison to the basic fragmentation I.a

The construction of the permutation subgraph can be considered as the reversed process of the *basic fragmentation (variation I.a)* see Section 3.3.2. Recall that the I.a fragmentation process is: select the an active vertex v u.a.r, v selects an active neighbour u u.a.r. Then v *absorbs* u, u is inactive and we connect a directed edge $e(u,v)$.

We draw some comparisons between the two processes by the example below.

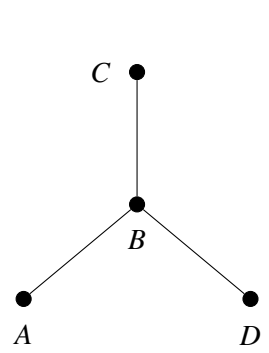
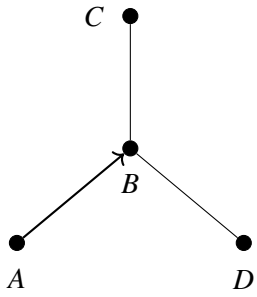
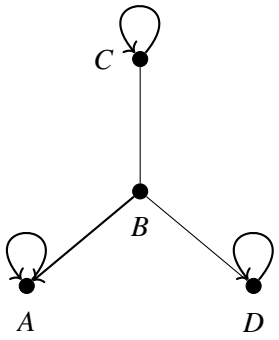
<table>
<tr><td>Input Graph</td><td colspan="4">First Iteration</td></tr>
<tr><td> 
</td><td colspan="2"> 
</td><td colspan="2"> 
</td></tr>
<tr><td>Desired outcome</td><td colspan="2">Any edge points to B</td><td colspan="2">Three roots at A,C,D</td></tr>
<tr><td>Algorithm</td><td>I.a</td><td>permSub</td><td>I.a</td><td>permSub</td></tr>
<tr><td>Main vertex</td><td>B</td><td>A or C or D</td><td>A or C or D</td><td>B</td></tr>
<tr><td>Probability</td><td>1/4</td><td>3/4</td><td>3/4</td><td>1/4</td></tr>
</table>

Table 4.1 Input: a *star* graph on 4 vertices with $\{B\}$ as its central vertex. The table illustrates the necessary selections for each algorithm to achieve the same desired outcome.

On the left: the desired outcome is to have an edge point to B. For *I.a*, it must be that B is the main vertex, B then absorbs any neighbour say A. The edge $e(A,B)$ is then connected. This occurs with probability $1/4$. On the other hand, for *permutation subgraph*, to achieve the desired outcome, it must be that either A,C,D is the main vertex, then the main vertex points to B with probability 1. Thus, this event occurs with probability $3/4$.

On the right: the desired outcome is to have B points to a vertex, hence resulting in **three roots** A,C,D. For I.a, this occurs when either A,C,D is selected as the main vertex. The main vertex then absorbs vertex B with probability 1. Hence, this event occurs with probability $3/4$. For permutation subgraph, this occurs only when B is the main vertex. Then B can point to any of remaining vertices, consequently generate three roots at A,C,D. This occurs with probability $1/4$.

In this sense, the permutation subgraph seems to be a reversed process of the I.a process.

4.1.3 The expected number of components of permutation subgraph

Consider the following observation

Proposition 6. *The number of components in H is the number of root vertices.*

From any vertex v which is not a root, we can traverse the path from v *forward* until we find the root. Each component can have only one root, because each vertex has out-degree 1. The number of roots is the number of components. □

We abbreviate the permutation subgraph as *permSub*.

4.1.3.1 The expected number of components of permSub of a cycle

Suppose the input cycle has n vertices. Since the order of inspection π is a permutation of V, there are $n!$ possibilities and each occurs with an equal probability of $1/n!$.

Fix a vertex v. Suppose its two neighbours are u and w. Notice that in order for v to be a root, its two neighbours have to be selected and become *inactive* prior to v's selection.

Assuming vertex v becomes *inactive* at the k^{th} step ($3 \leq k \leq n$), then u and w must be inspected at some step prior to k. Excluding u and w, there remains $k-2$ vertices to be removed (become inactive) before k, the total number of possibilities is $\binom{n-3}{k-3}$, and the number of arrangements is $\binom{n-3}{k-3}(k-1)!$. Excluding v, there are $n-k$ vertices after the k^{th} index with $(n-k)!$ possible arrangements. Hence, the total number of possibilities is

$$\begin{aligned}\binom{n-3}{k-3}(k-1)!(n-k)! &= \frac{(n-3)!}{(k-3)!(n-k)!}(k-1)!(n-k)! \\ &= (n-3)!(k-1)(k-2)\end{aligned}$$

for a fixed k^{th} position. Since $3 \leq k \leq n$:

$$\begin{aligned}\sum_{k=3}^{n}(n-3)!(k-1)(k-2) &= (n-3)!\sum_{k=3}^{n}(k-1)(k-2) \\ &= (n-3)!\frac{(n-2)(n-1)n}{3} \\ &= \frac{n!}{3}\end{aligned}$$

Therefore, the probability for attaching a self-edge to a vertex v is $Pr(v) = n!/3.n! = 1/3$. As there are n vertices, the expected number of roots is

$$E[\#\ roots] = n/3$$

□

4.1.4 The expected number of components of permSub of a r-regular graph

A cycle is a 2-regular graph. The previous strategy can be extended to the general case of r-regular graphs.

Suppose the input graph is *r-regular* i.e. $d(v) = r : \forall v \in V$. Fix a vertex v, assuming v becomes inactive at the k^{th} step, then $r+1 \leq k \leq n$. In order for v to attach a self-edge, all of v's neighbours must become inactive at some steps prior to k. As v has r neighbours, we need to fill in $k-r-1$ other positions from the population of $n-r-1$. Thus there are $\binom{n-r-1}{k-r-1}$ possibilities with $\binom{n-r-1}{k-r-1}(k-1)!$ arrangements. Then there are $(n-k)$ vertices to the right of k^{th}, with $(n-k)!$ possible arrangements. Altogether, we have the total number of possibilites for a fixed k^{th}

$$\begin{aligned}\binom{n-r-1}{k-r-1}(k-1)!(n-k)! &= \frac{(n-r-1)!}{(k-r-1)!(n-k)!}(k-1)!(n-k)! \\ &= (n-r-1)!(k-1)_r\end{aligned}$$

where $(k-1)_r = (k-1)\{(k-1)-1\}\ldots\{(k-1)-r+1\}$. Since $r+1 \leq k \leq n$, we have the total

$$\sum_{k=r+1}^{n}(n-r-1)!(k-1)_r = (n-r-1)!\sum_{k=r+1}^{n}(k-1)_r \tag{4.1}$$

To calculate the sum on the right, let's revert back one step, to the form

$$\sum_{k=r+1}^{n}(k-1)_r = \sum_{k=r+1}^{n}\frac{(k-1)!}{(k-1-r)!} = \frac{r!}{0!}+\frac{(r+1)!}{1!}+\frac{(r+2)!}{2!}+\cdots+\frac{(n-1)!}{(n-1-r)!}$$

Denote the sum on the right hand side by S, we have

$$\begin{aligned}S &= \frac{r!}{0!}+\frac{(r+1)!}{1!}+\frac{(r+2)!}{2!}+\ldots\frac{n!}{(n-r)!}, \\ \frac{S}{r!} &= \frac{r!}{r!.0!}+\frac{(r+1)!}{r.!1!}+\frac{(r+2)!}{r!.2!}+\ldots\frac{(n-1)!}{r!.(n-1-r)!} \\ &= \binom{r}{0}+\binom{r+1}{1}+\binom{r+2}{2}+\ldots\binom{r+(n-r)}{r} = \sum_{i=0}^{n-r}\binom{r+i}{i}\end{aligned}$$

Note that $\sum_{k=0}^{m}\binom{r+k}{k} = \binom{r+m+1}{m}$, this is because

$$\sum_{k=0}^{m+1}\binom{r+k}{k} = \sum_{k=0}^{m}\binom{r+k}{k}+\binom{r+m+1}{m+1} = \binom{r+m+1}{m}+\binom{r+m+1}{m+1} = \binom{r+m+2}{m+1}$$

by the Pascal's rule. And hence, we have

$$\sum_{i=0}^{n-r}\binom{r+i}{i} = \binom{r+(n-1-r)+1}{(n-1-r)} = \binom{n}{n-1-r} \tag{4.2}$$

Thus

$$S = \frac{n!.r!}{(n-1-r)!(r+1)} = \frac{n!}{(n-1-r)!(r+1)} \tag{4.3}$$

Replace this into (4.1) gives

$$(n-r-1)!\sum_{k=r+1}^{n}(k-1)_r = (n-r-1)!\frac{n!}{(n-1-r)!(r+1)} = \frac{n!}{r+1} \tag{4.4}$$

The probability for attaching a self-edge to a given vertex v is therefore

$$Pr(v \text{ is a root}) = \frac{n!}{(r+1)n!} = \frac{1}{r+1}. \tag{4.5}$$

It is seen that equation (4.5) applies to any vertex v with degree r. Hence we have that

Lemma 1. *In the permutation subgraph model, the probability that a vertex v with degree $d(v)$ becomes a root is*

$$Pr(v \text{ is a root}) = 1/\{d(v)+1\}.$$

□

In a r-regular graph, as there are n vertices with equal degree r, the expected number of components is therefore $E[\#\text{ components}] = n/(r+1)$. □

4.2 Related models - random mapping graph, 1-out graph

In this section, we discuss some other random graph models which share similarities with the permutation subgraph. The models are: *random mapping graph* and *k-out graph*. These models have been studied in great details by e.g. Bollobas [6], Frieze [22]. Our intention here is to introduce the models and compute some properties to compare them with the permutation subgraph.

A *random mapping graph* (r.m.g) is constructed by having each vertex *map* to one of its neighbour or itself. The main difference, compared to permutation subgraph (permSub), is that the *inactive* concept is absent i.e. every vertex is available throughout the computation. One can imagine that the process can be done synchronously, with every vertex independently maps to a chosen vertex. The result r.m.g is naturally a directed graph, every vertex has out-degree 1. Further, every component in r.m.g contains exactly one *loop*, *double-edge* or directed cycle of length at least 3. We see that it is not possible in to produce a directed cycle of length at least 2 in the permSub graph.

The k-out graph model generalises random mappings. Let $G_{k\text{-out}}$ be a random directed graph, with its edges constructed by independently connecting each vertex $v \in V$ to exactly k neighbours i.e. each of the $\binom{d(v)}{k}$ possibilities occurs with equal probability. Similar to the r.m.g, $G_{k\text{-out}}$ can be constructed synchronously, mapping every vertex to its k neighbours. In this sense, the p.s model is different since it is constructed iteratively.

If $k = 1$, then the number of r.m.g is very similar to the number of $G_{1\text{-out}}$. In fact, we see that $G_{1\text{-out}}$ is r.m.g without loops. It is shown in [6] that most results concerning the $G_{1\text{-out}}$ transfer to r.m.g and vice versa. Hence we choose the 1-out graph to generalise and compare with permSub in the sections below.

4.2.1 The expected number of components of $G_{\text{1-out}}$

To compare with the results of permutation subgraph in Section 4.1.3, we show that

Lemma 2. *The expected number of components of* $G_{1\text{-}out}$

- *of a cycle is*

$$E[\#\ components] = n/4 + 1/2^{n-1}.$$

- *of a r-regular graph is*

$$E[\#\ components] = \frac{n}{2r} + \log(r) - \frac{r-1}{r} - \frac{1}{2}\Big(\frac{r-1}{r}\Big)^2.$$

Denote by $\mathscr{G}$ the collection of all possible $G_{\text{1-out}}$ that can be constructed from an input graph G. From its definition, the total number of graphs in $\mathscr{G}$ is therefore

$$|\mathscr{G}| = \prod_{v \in V} d(v).$$

4.2.2 The expected number of components of $G_{\text{1-out}}$ of a cycle

Since for each $v \in V$: $d(v) = 2$ hence $|\mathscr{G}| = 2^n$. Furthermore, if edges are sampled with the uniform distribution then the probability of one particular $G \in \mathscr{G}$ is $1/2^n$.

As pointed out, each component in $G_{\text{1-out}}$ is unicyclic that is formed by a double-edge. Further, for a pair of vertices (u,v) a double-edge is added with probability $Pr = 1/4$. As there are n edges, the expected number of double-edge is therefore $n/4$. The only other probability is a directed cycle oriented clockwise or anti-clockwise, an event of probability $2/2^n$. Thus

$$E[\#\ \text{components}] = n/4 + 1/2^{n-1}.$$

□

4.2.3 The expected number of components of $G_{\text{1-out}}$ of a r-regular graph

In this section, we examine the $G_{\text{1-out}}$ of a *r-regular-graph* constructed using the *configuration model* [6]. Let $G(V,E)$ be an undirected r-regular graph with $V = \{1,2\ldots n\}$; $d_i = r, (\forall i \in V)$ and $\sum_i d_i = 2m = rn$. Let $W = \bigcup_{i=1}^{n} W_i$ be a set of $2m$ labelled vertices $|W_i| = d_i$, each W can be considered as a *cell* of $v \in V$, and for each labelled vertex x, if $x \in W_i$ then x assumes vertex label v_i. A *configuration* F is a partition of W into m pairs of *vertices*. Then, a (multi)graph $\gamma(F)$ is formed with n vertices and taking pairs in F as edges.

A picturesque definition for the *configuration model* is as follows. For each vertex $v \in V$, we chop its edges in half, leaving a number $d(v)$ (or r in this case) half-edges or stubs. In total, there are

$\sum_i d(i) = 2m = rn$ stubs. We then connect the stubs in pairs, selecting each uniformly at random. The result at the end is a *(multi)graph* as *loops* and *double-edges* are allowed.

We are interested in the number connected components of $G_{\text{1-out}}$ of a simple *r-regular-graph* G. Let γ be the number of cycles in $G_{\text{1-out}}$, since the components are unicyclic, γ is also the number of components. Let γ_k be the number of cycles of length k. Then the number of loops or self-edges is $\gamma_1 = 0$. The probability of a double-edge (cycle of length 2) is clearly $1/r^2$, as there are $rn/2$ edges, $E[\gamma_2] = n/2r$.

In the following section, we calculate the number of cycles length k for $3 \leq k \leq n$ in $G_{\text{1-out}}$. Clearly there are

$$\binom{2m}{2}\binom{2m-2}{2}\binom{2m-4}{2}\cdots\binom{2}{2}$$

ways to pick the pairs in W. However, as m edges can be rearranged in $m!$ ways, the number of possibilities is therefore

$$N(m) = \frac{\binom{2m}{2}\binom{2m-2}{2}\binom{2m-4}{2}\cdots\binom{2}{2}}{m!} = \frac{(2m)!}{m!.2^m} = (2m-1)(2m-3)\ldots(1) = (2m)!! \tag{4.6}$$

Assume we fix a number of k edges among the m edges, then there are

$$N(m-k) = \frac{\binom{2m-2k}{2}\binom{2m-2k-2}{2}\binom{2m-2k-4}{2}\cdots\binom{2}{2}}{(m-k)!} = \frac{(2m-2k)!}{(m-k)!.2^{(m-k)}} = (2m-2k)!!$$

configurations containing these k edges. This can be rewritten as

$$\begin{aligned} N(m-k) = \frac{(2m-2k)!}{(m-k)!.2^{(m-k)}} &= \frac{(2m)!}{m!.2^m}.\frac{(m-k+1)(m-k+2)\ldots(m).2^k}{(2m-2k+1)(2m-2k+2)\ldots(2m)} \\ &= N(m).\frac{1}{(2m-1)(2m-3)\ldots(2m-2k+1)} \end{aligned}$$

Hence, the probability that k edges appear in a configuration is

$$Pr(k) = \frac{N(m-k)}{N(m)} = \frac{1}{(2m-1)(2m-3)\ldots(2m-2k+1)} \tag{4.7}$$

Let us calculate the number of oriented cycles of length k in the configuration model. A cycle of length k can be defined as a set of k edges i.e. $\{e_1, e_2, \ldots e_k\}$, that connect k distinct *cells*. That is, for k distinct cells i.e. $W_i, W_{i+1}, \ldots W_{i+k-1}$, the edge e_i connects a pair of cells (W_i, W_{i+1}) (with $W_{i+k} = W_i$). There are $\binom{n}{k}$ ways to choose k cells, with $k!$ possible arrangements. To connect each pair of cells, notice that for each unconnected *cell* there are r available stubs, after it is connected there remains $(r-1)$ stubs to connect that cell to the next. Thus, there are $(r(r-1))^k$ possibilities. Hence, there are $\binom{n}{k}(k!)(r(r-1))^k$ number of oriented cycles. As there are k starting points and 2 directions

for each oriented cycle, the total number of k-cycles is reduced by this factor:

$$\binom{n}{k}(k!)(r(r-1))^k/2k = \binom{n}{k}.\frac{(k-1)!}{2}.\binom{r}{2}^k.2^k \quad \text{or} \quad (n)_k.(r(r-1))^k/2k \tag{4.8}$$

Then, the expected number of k-cycles in a configuration is

$$\begin{aligned} E[k] &= \frac{(n)_k.(r(r-1))^k}{(2m-1)(2m-3)\dots(2m-2k+1).2k} \\ &= \frac{rn}{(rn-1)}.\frac{r(n-1)}{(rn-3)}.\frac{r(n-2)}{(rn-5)}.\dots.\frac{r(n-k+1)}{(rn-2k+1)}.\frac{(r-1)^k}{2k} \end{aligned}$$

A crude approximation can be that $r^k.(n)_k < (rn)^k$, and $(rn-1)(rn-3)\dots(rn-2k+1) < (rn)^k$. Which yields $E[k] \approx \frac{(r-1)^k}{2k}$, for a fixed k. Finally, in order to form a k-cycle in $G_{\text{1-out}}$, each vertex has to select others with $Pr = 1/r$, hence a k-cycle formed occurs with probability $Pr = (1/r)^k$. Note that for each k-cycle there are 2 ways to orient it, hence altogether:

$$E[\gamma_k] \approx 2\frac{(r-1)^k}{2k}.\frac{1}{r^k} = \frac{1}{k}\Big(\frac{r-1}{r}\Big)^k$$

Therefore, the number of cycles in $G_{\text{1-out}}$ is:

$$E[\gamma] = \frac{n}{2r} + \sum_{k\geq 3}\frac{1}{k}\Big(\frac{r-1}{r}\Big)^k \tag{4.9}$$

For the second term, put $(r-1)/r = z$ which gives $\sum z^k/k = \log\frac{1}{1-z}$. Thus:

$$\sum_{k\geq 3}\frac{1}{k}\Big(\frac{r-1}{r}\Big)^k = \log\frac{1}{1-\frac{r-1}{r}} - \frac{r-1}{r} - \frac{1}{2}\Big(\frac{r-1}{r}\Big)^2 \tag{4.10}$$

Replace (4.10) into (4.9):

$$E[\gamma] = \frac{n}{2r} + \log(r) - \frac{r-1}{r} - \frac{1}{2}\Big(\frac{r-1}{r}\Big)^2 \tag{4.11}$$

□

4.2.3.1 Remark

There is also an apparent difference between r.m.g, $G_{\text{1-out}}$ and fragmentation processes when the input graph is a complete graph $G = K_n$. In which case, the expected number of components of permSub of G is 1 i.e. there is exactly one root, which is the last vertex in the permutation π. Also, the subgraph is a random Hamiltonian path of G. Furthermore, any variation of fragmentation process that were studied in previous section, the number of components is also 1 i.e. the root is always found in the last iteration.

On the other hand, consider simple graphs produced by r.m.g and $G_{1\text{-out}}$. Bollobas [6] shows that the expected number of components of r.m.g of a K_n graph is $\sim \log n$. Further, $G_{1\text{-out}}$ is a loopless r.m.g, its expected number of components is: $\log n - E[\#\text{ r.m.g without loop}]$. The number of r.m.g without loop is the number of derangements of a permutation. This is a well known quantity, the expected number of derangements is $\sim 1/e$. Thus, the expected number of components in $G_{1\text{-out}}$ is $\sim \log(n/e^{1/e}) \approx \log(n/1.444)$.

4.3 Conclusion

In this chapter, we introduced the *permutation subgraph* model - a graph fragmentation variation. This graph model shares some similarities with the basic fragmentation model (I.a). While the latter seems problematic to study (as shown in Chapter 3, Section 3.3), it seems more promising to obtain analytical results with the former. Thus, we analysed a basic property: *the expected number of components* of the permutation subgraph model of various graphs, including: a cycle, r-regular-graph and complete graph. The results are then compared with those of some known models which are 1-out graph and random mapping graph.

Chapter 5

Permutation subgraph of random graph $G(n,p)$

In this section, we study the *permutation subgraph* of the random graph $G(n,p)$. The permutation subgraph was introduced in Section 4.1. Its core procedure is: generate a random permutation $\pi = (v_1, \ldots, v_n)$ of the vertex set. We visit each vertex $v_i \in \pi$ in order and connect v_i to a *random neighbour* v_j where $i < j$ i.e. v_j succeeds v_i in π.

We modify the original process. For each vertex v_i, instead of *selecting a random succeeding neighbour*, we connect v_i to **first succeeding** neighbour v_j. That is we connect the edge (v_i, v_j) if and only if $i < j$ and there exists $(v_i, v_j) \in E$ and there is no edge (v_i, v_k) for $k < j$. In this chapter, we analyse the following properties of the model in each corresponding section

Section 2: The existence of a giant component;

Section 3: The expected number of components;

Section 4: The location of the root vertices;

Section 5: The probability that the permutation subgraph is connected;

Section 6: The expected path length;

Section 7: The limiting distribution for the size of the left subtree of a vertex.

We begin by introducing the model.

5.1 Definition

Let $G(V,E)$ be a simple, undirected graph with $|V| = n$; let $\pi = \{v_1, v_2 \ldots, v_n\}$ be a random permutation of V. We denote vertices in π as v_i, v_j where i, j are its indices in π. A *permutation subgraph* H is generated as follows. For each vertex v_i, we direct a single *forward* edge from v_i to some vertex v_j

i.e. (v_i, v_j), if and only if $j > i$ and there exists an edge $(v_i, v_j) \in E(G)$ and no edge (v_i, v_k) for $k < j$. If, for any vertex v_i, there is no edge between v_i and every vertex that succeeds it in π, then v_i *fails to connect*, and regarded as a *root* vertex. Thus, we have

Proposition 7. *The number of components in H is exactly the number of root vertices. Further, each component is a tree rooted at such a vertex.*

In this section, we consider the graph G is generated by the random graph $G(n,p)$ model. Thus, for every pair of vertices $u, w \in V(G)$, the probability

$$p = \Pr(\text{there is an edge between } u \text{ and } w),$$
$$1 - p = \Pr(\text{there is no edge between } u \text{ and } w).$$

We prove this transfers directly to H, in such sense that for any vertex v_i, the probability that the edge (v_i, v_{i+1}) is *included* is p, and vice versa for the complementary event.

Among π, let us select a vertex v_i. From v_i we can make a *jump* to some vertex v_j $(i < j)$ with probability p; then from v_j to v_k and so on, in the end we have created a path of an arbitrary length. This is proved in the following proposition.

Proposition 8. *Let there be a path P_i, starting from vertex v_i. For any $j > i$, the probability that vertex v_j is in P_i is exactly p, independently from other vertex.*

To construct P_i, we repeat the following procedure. Starting from vertex v_i, we examine each vertex v_k $(i < k \leq n)$ *in order*. Suppose we have reached a vertex v_{j-1}. Now, there are only two cases: either v_{j-1} is *in* P_i or it *is not* . In the former case, v_j would be in P_i by including the edge (v_{j-1}, v_j), with probability p. In the latter, a vertex v_l $(i \leq l < j-1)$ was the last vertex of P_i, and the edges $(v_l, v_{l+1}) \ldots (v_l, v_{j-1})$ are failed to be added. That v_l is still being *queried* for connection. In which case, the edge (v_l, v_j) is added with probability p.

In either case, v_j is added to the path P_i with probability p independently of the previous history of the path construction.

This proves our claim. □

5.2 The giant component

Theorem 1. *Let $\varepsilon > 0$. For all $\delta > 0$ if $p > \frac{(1+\delta)\sqrt{\log n}}{\sqrt{\varepsilon n}}$, then for all $1 \leq i \leq (1-\varepsilon)n$, each v_i is in the same component.*

We divide the permutation into two segments: the beginning Ω' and the ending Ω, in which there are $|\Omega'| = (1-\varepsilon)n$, $|\Omega| = \varepsilon n$ $(0 < \varepsilon < 1)$ vertices, respectively.

Let us construct the path P_1 starting from vertex v_1, we have shown that for any $j > i$, the probability that $v_j \in P_1$ is p. As P_1 spans across the entire the ordering π, one can expect many

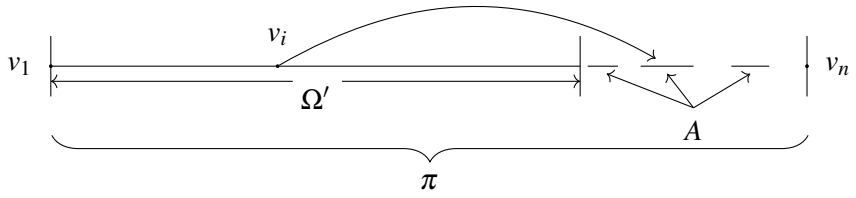


Fig. 5.1 The giant component of a permutation subgraph.

vertices to connect *directly* to P_1 as each vertex is sequentially inspected. If we look at the ending segment, we see that

Lemma 3. *For $p \geq (1+\delta)\sqrt{\log n}/\sqrt{\varepsilon n}$, with high probability, there are at least $A = \sqrt{\varepsilon n \log n}$ vertices in Ω, which are all lying in the path P_1.*

Consider the ending segment Ω, consisting of εn vertices. Let X_j be an indicator random variable that assumes the value 1 if vertex $v_j \in \Omega$ is in the path P_1 and 0 otherwise, thus $E[X_j] = p$. Furthermore, let X be their sum, for all vertices in Ω we get

$$X = \sum_{j \in \Omega} E[X_j] = \varepsilon np$$

Using Chernoff bound (see Theorem B.2), substituting $\mu = \varepsilon np$ and $1 - \theta = \frac{1}{(1+\delta)}$ implying $\theta = \frac{\delta}{1+\delta}$, we get

$$\Pr(X \leq \frac{\varepsilon np}{1+\delta}) \leq \exp\Big\{ - \frac{\delta^2}{2(1+\delta)^2} \varepsilon np \Big\}$$

and if $p > \frac{(1+\delta)\sqrt{\log n}}{\sqrt{\varepsilon n}}$, replacing in we get

$$\Pr(X \leq \sqrt{\varepsilon n \log n}) \leq \exp\Big\{ - \frac{\delta^2 \sqrt{\varepsilon n \log n}}{2(1+\delta)} \Big\}$$

For $\delta > 0$, $\exp\Big\{ - \frac{\delta^2 \sqrt{\varepsilon n \log n}}{2(1+\delta)} \Big\} \to 0$ when $n \to \infty$, we can conclude that with high probability $X \geq \sqrt{\varepsilon n \log n}$. Thus, it is implied that there are at least $A = \sqrt{\varepsilon n \log n}$ vertices, located within Ω, that are all connected to P_1. □

Looking back at the beginning segment Ω', for those vertices which have not connected *directly* to P_1, each still has a *chance* to *join* P_1 *indirectly* by connecting to those in A. We next prove that

Lemma 4. *For any vertex $v_i \in \Omega'$, with high probability, v_i is connected to a vertex in A.*

Suppose we have constructed the path P_i, which starts from vertex v_i for $1 < i \leq (1-\varepsilon)n$, up to the end of Ω'. Now we begin the construction of P_i in the last segment Ω. Previously, we showed that in Ω, there are at least A vertices which all lie in the path P_1. Hence, using Proposition 8, with

probability p, the path P_i will connect to the first vertex of A; thus connecting the path P_i to P_1. The probability that the path P_i *does not connect* to any vertex of A, let this be event R, is

$$\Pr(R) = (1-p)^A \leq e^{-pA} \leq e^{-(1+\delta)\log n} = \frac{1}{n^{1+\delta}} \tag{5.1}$$

since A is shown to be at least $A = \sqrt{\varepsilon n \log n}$ in Lemma 3. The inequality used is $(1-x) \leq e^{-x}$ $(0 < x < 1)$. Taking logs of both side and using Taylor expansion for the left hand side yields $-x - x^2/2 - x^3/3 - \cdots \leq -x$.

So with high probability, the path starting from the vertex v_i will connect to the path from v_1. We can repeat this process as long as there remain vertices which are not within the component we have so far constructed. As at most $|\Omega'| \leq n$ remain, the probability that any vertex is not connected is

$$|\Omega'| \frac{1}{n^{1+\delta}} \leq n^{-\delta}$$

and so we see with probability $1 - O(n^{-\delta})$ all of the first Ω' vertices in the ordering are connected to v_1 as required. □

From Lemma 3 and Lemma 4, there exists a component of size at least $f(n) = (1-\varepsilon)n + \sqrt{\varepsilon n \log n}$ which proves Theorem 1. □

5.2.1 Lower bound

Theorem 2. *If $p \leq 1/\omega\sqrt{n}$, where $\omega \leq \sqrt{n}/30$, then the maximum component size of the permutation subgraph of $G(n,p)$ is $O(n/\sqrt{\omega})$ w.h.p.*

For $s < t$ let $F(s,t)$ be the event that the paths starting at vertices s,t meet at some vertex $t \leq v \leq n$. Then the probability of the complementary event $\overline{F}(s,t)$ is

$$\Pr(\overline{F}(s,t)) = (1-p)(1-p^2)^{n-t-1}$$

This occurs because the path from s does not include vertex t and both of the two paths do not touch some vertex $v > t$ thereafter.

Consider the following inequality

$$(1-x)^t \geq 1 - tx \qquad (0 \leq x \leq 1) \tag{5.2}$$

It is seen (5.2) is true for $x = 0$. Furthermore, consider

$$t = 1 + \cdots + 1 \geq 1 + (1-x) + (1-x)^2 + \cdots + (1-x)^{t-1} = \frac{1-(1-x)^t}{1-(1-x)}$$

thus $xt \geq 1 - (1-x)^t$ for $0 < x \leq 1$, which yields (5.2).

Consider the probability of the event $F(s,t)$

$$\begin{aligned}\Pr(F(s,t)) = 1-(1-p)(1-p^2)^{n-t-1} &\leq 1-(1-p)[1-p^2(n-t-1)]\\ &\leq (n-t-1)p^2(1-p)+p\\ &\leq (n-t)p^2+p\end{aligned}$$

Let $X = |\{(s,t) : \text{paths meet}\}|$ then

$$\begin{aligned}E[X] &= \sum_{s=1}^{n}\sum_{t>s}\Pr(F(s,t))\\ &\leq \sum_{s=1}^{n}\sum_{t>s}((n-t)p^2+p)\\ &\leq n^2p+p^2\sum_{s=1}^{n}\sum_{L=1}^{n-s}L\\ &\leq n^2p+p^2/2\sum_{S=1}^{n}S^2 = \frac{p^2n^3}{6}+n^2p\end{aligned}$$

Let $p = 1/\omega\sqrt{n}$, then provided $\omega \leq \sqrt{n}/30$, $E[X] \leq n^2/(5\omega^2)$, and $\Pr(X \geq \omega E[X]) \leq 1/\omega$. Thus w.h.p. $X \leq n^2/(5\omega)$.

Consider a path $P = \{1,2,3\}$, then the events $F(1,2)$, $F(1,3)$ and $F(2,3)$ have occurred. Generally, given a path of length k, then to form such path there are $\binom{k}{2} \approx k^2$ (if k large) paths meet. Therefore, if some component is size at least $n/\sqrt{\omega}$ then at least n^2/ω paths meet. A contradiction. □

5.3 Expected number of components

Theorem 3. *The expected number of components of a permutation subgraph of a random graph $G(n,p)$ is $\frac{1}{p}(1-(1-p)^n)$.*

In previous section, we showed that

Lemma 5. *For a given vertex v with degree $d(v)$, the probability that v becomes a root is*

$$Pr(v \text{ is a root}) = 1/\{d(v)+1\}$$

Proof. See Section 4.1.4. □

For every vertex v in a $G(n,p)$ graph

$$Pr\{d(v)=k\} = \binom{n-1}{k}p^k(1-p)^{n-1-k}$$

Denote by X_k the number of vertices with degree k, then

$$E[X_k] = n\binom{n-1}{k}p^k(1-p)^{n-1-k}$$

By Lemma 5, a vertex with degree k becomes a root with probability $1/(k+1)$. Hence, let Y be the expected number of root vertices. Then

$$E[Y] = \sum_{k=1}^{n-1}\frac{E[X_k]}{k+1} = n\sum_{k=0}^{n-1}\frac{1}{k+1}\binom{n-1}{k}p^k q^{n-1-k}$$

Let $f(n)$ denote the sum above and let $F(n)$ be its ordinary generating function (OGF). Multiply by x^n and summing on $n \geq 0$ we have

$$\begin{aligned}
F(x) &= \sum_n x^n n \sum_k \frac{1}{k+1}\binom{n-1}{k}p^k q^{n-1-k} \\
&= \sum_k \sum_n \frac{n}{k+1}\binom{n-1}{k}p^k q^{n-1-k}x^n \\
&= \sum_k p^k q^{-(k+1)} \sum_n \binom{n}{k+1}(qx)^n \\
&= \sum_k p^k q^{-(k+1)}\Big[\frac{(qx)^{k+1}}{(1-qx)^{k+2}}\Big] \qquad \Big(\text{by: } \sum_n \binom{n}{k}z^n = \frac{z^k}{(1-z)^{k+1}}\Big) \\
&= \frac{x}{(1-qx)^2}\sum_k \frac{(px)^k}{(1-qx)^k} \\
&= \frac{x}{(1-qx)^2}\sum_k \Big(\frac{px}{1-qx}\Big)^k
\end{aligned}$$

Substitue $r = \frac{px}{1-qx}$ then the summand is $\sum_k r^k = \frac{1}{1-r}$. Therefore:

$$F(x) = \frac{x}{(1-qx)^2}\frac{1}{1-\frac{px}{1-qx}} = \frac{x}{(1-qx)}\frac{1}{1-x(p+q)} = \frac{x}{(1-qx)}\frac{1}{1-x} = \sum_n (\sum_{0\leq k\leq n-1} q^k)x^n$$

Therefore

$$E[Y] = [x^n]F(x) = \sum_{0\leq k\leq n-1} q^k = \frac{1-q^n}{1-q} = \frac{1}{p}\{1-(1-p)^n\}$$

□

5.4 The root segment Ω

Let us select a vertex v, and assume there are exactly ℓ vertices which are placed after v in the ordering π. We denote the segment consisting of these ℓ vertices by Ω, and the remaining vertices as Ω'. Now

the event that v is a root (or equivalently, fails to connect to any vertex that succeeds it) is

$$Pr(v \text{ is root}) = (1-p)^{\ell} \cong e^{-\ell p}$$

when $p \to 0$.

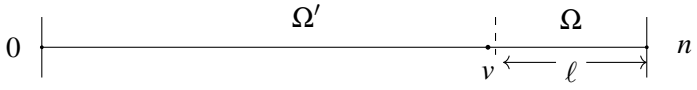


Fig. 5.2 Root segment of a permutation subgraph.

Lemma 6. *If $p \to 0$ but $p \geq \log n/n$, then w.h.p for any $\alpha > 1$ all roots are located in the final segment of length at most $\alpha \frac{1}{p} \log \frac{1}{p}$.*

To prove the above lemma, we show that with high probability, there is no *root* in the beginning segment $\Omega' = [1, \ldots, n - \alpha \frac{1}{p} \log \frac{1}{p}]$.

For any vertex $v \in \Omega'$, the probability that v is *a root* is $(1-p)^{\ell}$, where ℓ is the number of vertices succeed v in π. Thus, the expected number of roots in the *beginning segment* Ω' is

$$\begin{aligned} E[\text{roots in } \Omega'] = \sum_{k=\ell}^{n-1} (1-p)^k &= (1-p)^{\ell} \frac{1-(1-p)^{n-\ell}}{1-(1-p)} \\ &\leq \frac{(1-p)^{\ell}}{p} \qquad (\text{as } 1-(1-p)^{n-\ell} \leq 1) \\ &\leq \frac{1}{p} e^{-\ell p} \qquad (\text{as } 1-x \leq e^{-x}) \end{aligned}$$

Let us choose

$$\ell = \alpha \frac{1}{p} \log \frac{1}{p} \quad (\alpha > 1)$$

However, $\ell \leq n$ must also hold; so by putting $p = \omega/n$, we can bound ω. Substituting p in the above equation, we get $\frac{1}{\omega}(\log n - \log \omega) \leq 1/\alpha$. With $\omega = \log n$, the inequality is simplified to $1 - \frac{\log \log n}{\log n} = 1 - o(1) \leq 1/\alpha$, thus α approaches 1 from above. On the contrary, assuming $\omega < \log n$ i.e. $\omega = \log n/a$ for some integer $a > 1$, then

$$\frac{\ell'}{n} = a\big(1 - \frac{\log \log n}{\log n} + \frac{\log a}{\log n}\big) = a(1 - o(1)) > 1$$

Thus, it must be that $\omega \geq \log n$.

The expected number of roots is

$$E[\text{roots in } \Omega'] \leq \frac{1}{p} e^{-\alpha \log \frac{1}{p}} = p^{\alpha - 1} \to 0$$

given that $p \to 0$. Or it might be the case that α is close to 1 that $\alpha - 1 \to 0$, in which case the expectation is bounded from the above by 1, which is still negligible. Thus given that we choose $\ell \geq \alpha \frac{1}{p} \log \frac{1}{p}$ w.h.p there would be no root in the $\Omega' = [1, \ldots, n-\ell]$ segment. □

5.5 Probability the subgraph H is connected

Theorem 4. *Let R be the event that the permutation subgraph H is connected. Then the probability that H is connected is given by*

$$Pr(R) = \prod_{\ell=1}^{n-1} (1 - (1-p)^{\ell}) \tag{5.3}$$

$Pr(R)$ is bounded by

$$p^{1/p} \leq Pr(R) \leq pe^{2-p-1/p}$$

For some ranges of p we have, more precisely,

1. *$p \to 0$ but $p \geq \omega/n$*
$$Pr(R) = (1 + o(1))e^{-\frac{1}{p}\pi^2/6}$$
2. *$p = c$ where c is constant, then $Pr(R)$ converges to a positive constant, which tends to*
$$Pr(R) = e^{-\pi^2/6c}$$
as $c \to 0$.
3. *$p \to 1$, $Pr(R) \to 1$.*

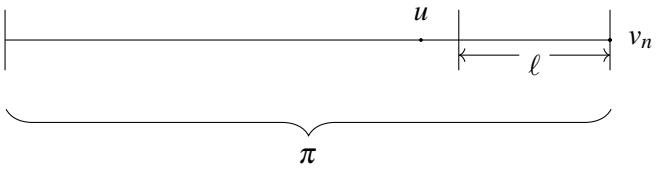


Fig. 5.3 Examining a permutation from the end vertex backward.

Let us select the last vertex v_n in the ordering π and examine each vertex to the left of v_n *in succession*. Observe that, in order for the current graph segment to be connected, each time a new vertex $v_{n-\ell}$ ($\ell = 1, 2, \ldots, n-1$) is *examined*, we need to connect $v_{n-\ell}$ to any of the *visited* vertices. For instance, v_{n-1} connects with v_n with probability $1 - (1-p) = p$; v_{n-2} connects with the current segment consisting of $\{v_{n-1}, v_n\}$ with probability $(1 - (1-p)^2)$ and so on.

Therefore, the probability of the event $\bar{A}_\ell$ that $v_{n-\ell}$ fails to connect to any of the succeeding vertices is: $Pr(\bar{A}_\ell) = (1-p)^{\ell}$. Hence the probability of the event A_ℓ that vertex $v_{n-\ell}$ connects to someone is $Pr(A_\ell) = (1 - (1-p)^{\ell})$.

To keep the graph H connected, we repeat this process by looking further left from v_n until all vertices have been inspected. Thus, in order for H to be connected, the probability is

$$Pr(R) = Pr(A_1 \text{ and } A_2 \text{ and } A_3 \ldots A_{n-1}) = \prod_{\ell=1}^{n-1}(1-(1-p)^{\ell})$$

Substitue $q = 1-p$ and take log of both side and raise to power of e, we get

$$Pr(R) = \exp\Big\{\sum_{\ell=1}^{n-1}\log(1-q^{\ell})\Big\} = \exp\Big\{-\sum_{\ell=1}^{n-1}\sum_{j=1}^{\infty}\frac{(q^{\ell})^j}{j}\Big\}$$

Denote the sum in the exponent by $S(q) = \sum_{\ell=1}^{n-1}\sum_{j=1}^{\infty}\frac{q^{\ell j}}{j}$, then $Pr(R) = e^{-S(q)}$. We next evaluate $S(q)$.

Switching the order of summation and using $\sum_{k=1}^{n-1}x^k = x(1-x^{n-1})/(1-x)$, we get:

$$S(q) = \sum_{j=1}^{\infty}\frac{1}{j}\frac{q^j(1-q^{j(n-1)})}{1-q^j}$$

and hence if n tends to infinity and $p \geq \omega/n$

$$\lim_{n\to\infty}S(q) = \lim_{n\to\infty}\sum_{j=1}^{\infty}\frac{1}{j}\frac{q^j(1-q^{j(n-1)})}{1-q^j} = (1-o(1))\sum_{j=1}^{\infty}\frac{1}{j}\frac{q^j}{1-q^j} \tag{5.4}$$

If we factor out the polynomial in the denominator in $S(q)$, we get

$$\begin{aligned} S(q) &= \sum_j \frac{1}{j}\frac{q^j}{(1-q)(1+q+q^2+\cdots+q^{j-1})} \\ &= \frac{1}{p}\sum_j \frac{1}{j}\frac{(1-p)^j}{(1+(1-p)+(1-p)^2+\ldots(1-p)^{j-1})} \end{aligned}$$

Thus, for $p \to 0$, $(1-p)^j \to 1$, and $S(q) \to \frac{1}{p}\sum_j \frac{1}{j^2} = \frac{1}{p}\pi^2/6$. And thus

$$Pr(R) = e^{-\frac{1}{p}\pi^2/6} \tag{5.5}$$

If $p \to 1$ then $S(q) \to 0$ thus $Pr(R) \to e^0 = 1$.

If p is given by some constant then is q, so let $q = c$ $(0 < c < 1)$. Consider the series given by $S(q)$ in (5.4), let a_n be its n^{th} coefficient, $a_n = \frac{1}{n}\frac{q^n}{1-q^n}$. A ratio test on $S(q)$ shows that

$$L = \lim_{n\to\infty}\frac{\frac{1}{n+1}\frac{c^{n+1}}{1-c^{n+1}}}{\frac{1}{n}\frac{c^n}{1-c^n}} = \lim_{n\to\infty}\frac{c(n+1)}{n}\Big(\frac{1-c^{n+1}}{1-c^n}\Big) = \lim_{n\to\infty}(c+\frac{c}{n})\Big(\frac{1-c^{n+1}}{1-c^n}\Big) = c < 1$$

thus $S(q)$ converges. □

5.5.1 Bounding $Pr(R)$

Let us revert back to the following form of $S(q)$

$$S(q) = \sum_{\ell=1}^{n-1} \sum_{j=1}^{\infty} \frac{q^{\ell j}}{j}$$

If we expand the sum by ℓ

$$\begin{array}{lccccccccccccc}
\ell = 1: & q & + & \frac{q^2}{2} & + & \frac{q^3}{3} & + & \frac{q^4}{4} & + & \frac{q^5}{5} & + & \frac{q^6}{6} & + & \dots \\
\ell = 2: & & & q^2 & & + & & \frac{q^4}{2} & & + & & \frac{q^6}{3} & + & \dots \\
\ell = 3: & & & & & q^3 & & & + & & & \frac{q^6}{2} & + & \dots \\
\ell = 4: & & & & & & & q^4 & & + & & & + & \dots \\
\ell = 5: & & & & & & & & & q^5 & & & + & \dots \\
\ell = 6: & & & & & & & & & & & q^6 & + & \dots
\end{array}$$

Taking the sum by the columns, we get

$$S(q) = q + q^2(1+\frac{1}{2}) + q^3(1+\frac{1}{3}) + q^4(1+\frac{1}{2}+\frac{1}{4}) + \dots$$

It can be seen that for the m^{th} coefficient of $S(q)$ such that $m = j \times \ell \leq n-1$ then the sum associated with $[q^m]S(q)$ is the sum of all positive divisors of m, since $j \in [1;\infty]$ thus every integer $m/j \leq n-1$ available for ℓ. Else for $m > n-1$ then clearly not all divisors of m are present.

Thus, if we take $n \to \infty$, we get

$$S(q) = \sum_{m=1}^{\infty} q^m \sum_{d|m} \frac{1}{d} \tag{5.6}$$

where $d|m$ denotes the sum over all positive divisors d of m. Let b_m be the m^{th} coefficient of $S(q)$, then b_m is bounded by

$$\frac{1}{m} \leq \sum_{d|m} \frac{1}{d} = b_m \leq \sum_{k=1}^{m} \frac{1}{m} = H_m \tag{5.7}$$

where H_m is the m^{th} Harmonic number. Thus

$$\sum_{m=1}^{\infty} \frac{q^m}{m} = \ln\big(\frac{1}{1-q}\big) \leq \sum_{m=1}^{\infty} b_m q^m \leq \sum_{m=1}^{\infty} H_m q^m = \frac{1}{1-q} \ln\big(\frac{1}{1-q}\big)$$

in which the generating function for the Harmonic number can be found in Wilf [63]. And since $1-q=p$, the inequality simplifies to

$$-\ln(p) \leq S(q) \leq -\ln(p)/p$$

Multiply by -1 and raise by e, we get

$$p \geq Pr(R) \geq p^{1/p} \tag{5.8}$$

The lower bound from (5.7) is rather crude. It can be seen that

$$b_m \begin{cases} = 1 & (\text{if } m = 1) \\ \geq 1 + 1/m & (\text{if } m \geq 2) \end{cases}$$

Thus

$$\sum_{m=1}^{\infty} b_m q^m \geq q + \sum_{m=2}^{\infty} (1 + \frac{1}{m}) q^m$$

which implies that

$$\begin{aligned} S(q) \geq q + \sum_{m=2}^{\infty} q^m + \sum_{m=2}^{\infty} \frac{q^m}{m} &= q + \frac{q^2}{1-q} - q + \ln\big(\frac{1}{1-q}\big) \\ &= \frac{q^2}{1-q} + \ln\big(\frac{1}{1-q}\big) \end{aligned}$$

And hence

$$Pr(R) = e^{-S(q)} \leq e^{-\frac{q^2}{1-q} + \ln(1-q)} = pe^{2-p-1/p} \tag{5.9}$$

which improves the upper bound for $Pr(R)$ in (5.8). □

5.6 Path length

In this section, we analyse the distribution of path lengths. It can be seen that, in the trivial case of $p = 1$, for every vertex v_i $(0 \leq i \leq n-1)$ the forward edge will be directed to its immediate successor (v_i, v_{i+1}) since the graph $G(n,p)$ is complete, each possible edge is present. The subgraph H is connected with probability 1 and is a Hamiltonian path of G.

Recall Proposition 8, which shows that for a path P starting from vertex v_i, for any vertex v_j $(i < j)$ the probability that $Pr(v_j \in P) = p$; using which we can calculate the expected length of P.

Lemma 7. *For any path P starting from a given vertex v, the probability of having exactly k other vertices joining P is equal to the probability $Pr(S_k)$ of having k successes in $L = n - v$ Bernoulli trials with probability p for success*

$$Pr(S_k) = p^k q^{L-k} \binom{L}{k}$$

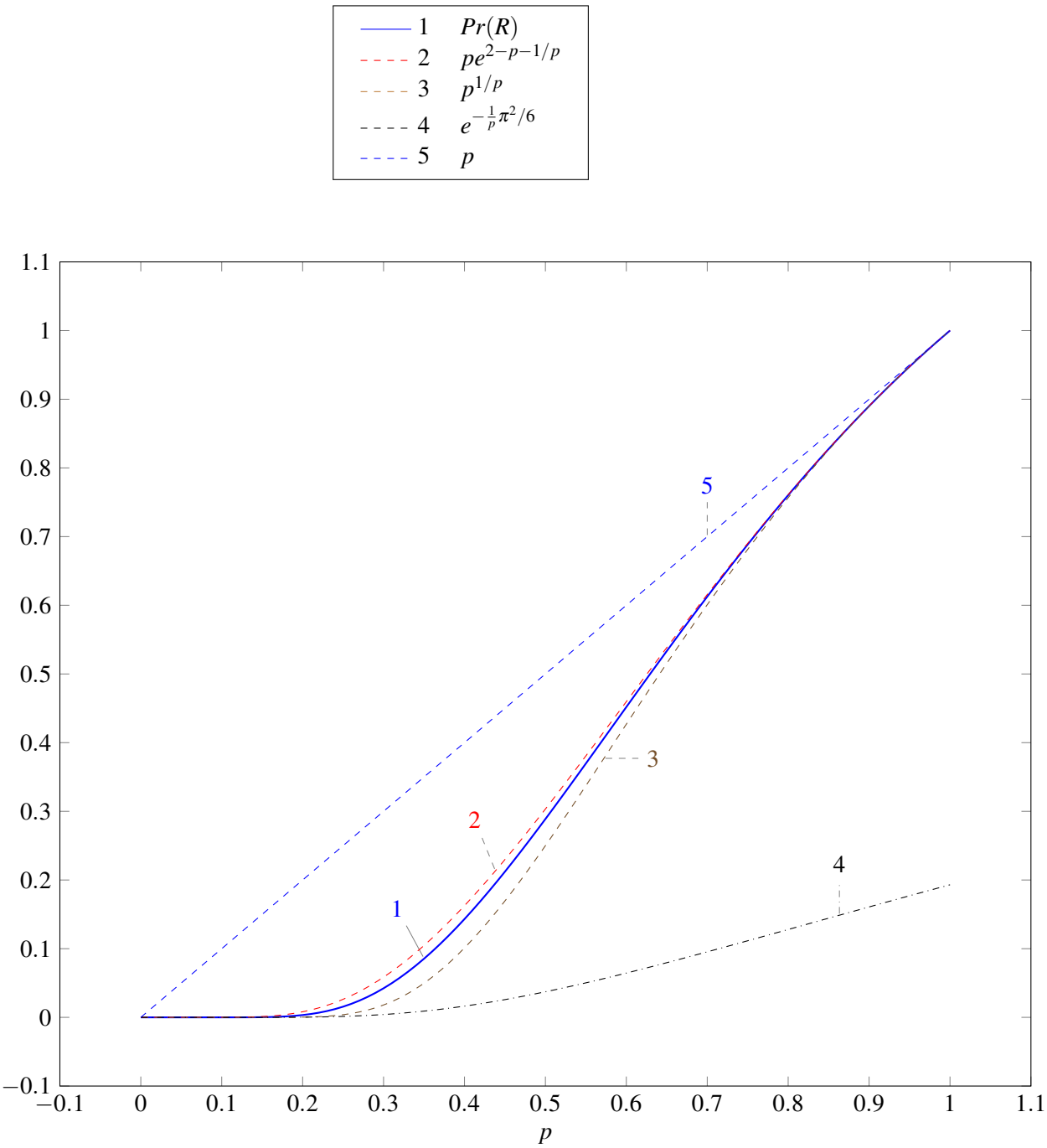
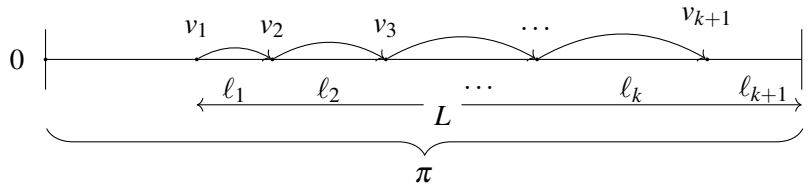


Fig. 5.4 The figure shows the numerical values of the probability $Pr(R)$ as a function of p. The line 1-blue shows the numerical values as given by equation (5.3); along with the two given bounds (2-red and 3-brown); and the approximation $e^{-\frac{1}{p}\pi^2/6}$ (4-black). It can be seen that, for approximately $p < 0.2$ the estimate given by $e^{-\frac{1}{p}\pi^2/6}$ is quite close. Further, for $p > 0.8$ the two bounds start to converge, which provides good approximation to $Pr(R)$.

Consequently, the average length of all paths starting from v $\overline{l_P}$ *is the mean of the binomial distribution*

$$\overline{l_P} = 1 + E[S_k] = 1 + Lp$$



We change our notation slightly. Denote by P_k^v a path of length k starting from vertex v. As we only examine vertices lying on a path, each vertex is labelled according to its order of joining the path i.e. $V(P) = \{v_1, v_2, \ldots, v_l\}$, in which v_1 is the starting vertex of $P_l^{v_1}$. Finally, let L be the number of vertices that succeed v_1 in the ordering π i.e. $L = n - v_1$.

To *jump* from some vertex v_i to v_{i+1}, clearly we have to *skipped over* some number of vertices. Let ℓ_i be this number, thus the probability that v_i connects to v_{i+1} is

$$Pr(\{v_i, v_{i+1}\}) = q^{\ell_i} p$$

Hence, the probability to each jump in P is: $q^{\ell_1} p, \ldots q^{\ell_k} p$, respectively; and finally $q^{\ell_{k+1}}$ to *cut-off* the path. Thus, we have

$$\begin{aligned} Pr(\{v_1, v_2\}, \ldots \{v_k, v_{k+1}\}) &= (q^{\ell_1} p)(q^{\ell_2} p) \ldots (q^{\ell_k} p) q^{\ell_{k+1}} \\ &= p^k q^{\ell_1 + \ell_2 + \cdots + \ell_{k+1}} = p^k q^{L-k} \end{aligned} \tag{5.10}$$

As the probability to make k jumps is equal, we need to know how many possibilities to make k-jumps? By counting the total number of vertices which have been *skipped over*, we have

$$\ell_1 + \ell_2 + \cdots + \ell_{k+1} = L - k \quad (\ell_i \geq 0)$$

The problem is analogous to the problem of distributing m number of indistinguishable balls into j number of distinguishable boxes. For which, there are $\binom{m+j-1}{j-1}$ possibilities (see Feller [19]). Thus, the number of possible paths P_k is: $\binom{(L-k)+(k+1)-1}{(k+1)-1} = \binom{L}{k}$. And therefore

$$Pr(P_k) = p^k q^{L-k} \binom{L}{k} \tag{5.11}$$

which is the binomial distribution of having k successes in L trials $b(k; L, p)$. □

The above discussion is rather lengthy. We can derive (5.11) directly from Proposition 8 by observing that since *the probability of joining a path* is *independent* for each vertex succeeds v_1, the sequence of vertex *connects/disconnects* [to P^v] is simply a sequence of L Bernoulli trials. Thus,

the probability of the path reaching length $k+1$ is equivalent to the probability of having exactly k successes in L Bernoulli trials, which is: $Pr(k) = p^k q^{L-k}\binom{L}{k} = b(k;L,p)$.

Thus expected number of vertices joining a path is the *mean of the binomial distribution*, thus the average length of a path, starting from a given vertex v is

$$\overline{l(P^v)} = 1 + E[S_k] = 1 + Lp$$

Further, as there are exactly n possible starting points for every path in H, the average path length is

$$\overline{l(\mathscr{P})} = \frac{1}{n}(1 + \sum_{v\in\pi} \overline{l(P^v)}) = \frac{1}{n}(1 + \sum_{L=0}^{n-1} Lp) = \frac{1}{n} + \frac{(n-1)p}{2} \approx \frac{np}{2} \tag{5.12}$$

From (5.10) and (5.11), the expected number of paths of length exactly $k+1$, starting with the first vertex in a segment of $L+1$ vertices inclusively, is

$$E[\textit{number of paths length } k+1 \textit{ in segment } L+1] = p^k q^{L-k}\binom{L}{k}$$

Thus, summing over all possible values for L, we get the expected number of paths of length exactly $k+1$ in H

$$E[\textit{number of paths length } k+1 \textit{ in } H] = \sum_{L=k}^{n-1} p^k q^{L-k}\binom{L}{k} \tag{5.13}$$

Substituting $L-k=j$ and $m=n-1-k$, the above sum can be rewritten as $\sum_{j=0}^{m} p^k q^j \binom{j+k}{k} = p^k \sum_{j=0}^{m} q^j \binom{j+k}{j}$, in which the summand is a partial sum of the negative binomial series $F(x) = \sum_{i=1}^{\infty}\binom{i+k}{i}x^i = \frac{1}{(1-x)^{k+1}}$.

5.7 Permutation subgraphs of infinite random graphs

In this section, we extend the random graph $G(n,p)$ to the set *integers*. For p constant, let

$$p = \Pr(\text{edge between integers } i,j)$$
$$1-p = \Pr(\text{no edge between integers } i,j)$$

Now, fix a vertex v, let there be the a vertex u which is inspected at position ℓ prior to v. Then, let A be the event that u attaches to v, then

$$\Pr(A) = p(1-p)^{\ell-1}$$

Thus the probability that u does not attach to v is:

$$\Pr(\overline{A}) = 1 - Pr(A) = 1 - p(1-p)^{\ell-1}$$

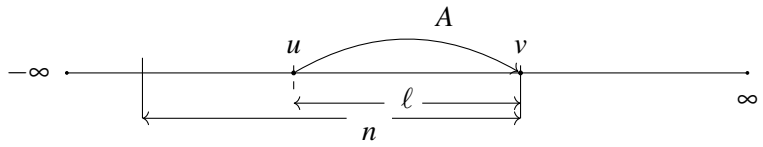


Fig. 5.5 Permutation subgraph of infinite random graph.

In order to give meaning to this model, for $p = p(n)$, we consider each vertex v has a *sliding window* of length n vertices to the *left*. Within which each edge uv $(u < v)$ exists with probability $p = p(n)$. The effect of restricting the graph this way is to take the product

$$\begin{aligned}\prod_{\ell=1}^{n}(1-p(1-p)^{\ell}) &= \exp(-p\sum_{\ell=1}^{n}(1-p)^{\ell}) = \exp\Big\{-p\big(\frac{1-(1-p)^{n+1}}{1-(1-p)}\big)\Big\} \\ &= \exp(-1+(1-p)^{n+1}) = e^{-1+o(1)}\end{aligned} \tag{5.14}$$

since $(1-p)^{n+1} \to e^{-np} = o(1)$, provided that $np \to \infty$ i.e. for $p > \omega/n$.

Inspired by this, we propose a limiting model on the integers, in which the number of left children of a vertex is Poisson with parameter λ, and the probability a vertex is a root i.e. attaches to no one on the right is $e^{-\lambda}$.

5.7.1 Main theorem

Theorem 5. *Consider the permutation subgraph on the integers as follows. Suppose each vertex v has a sliding window of length n vertices to its left. Such that within the window, each edge uv $(u < v)$ exists with probability p, i.e. u inserts the front edge uv with probability p. In such case, u is regarded as a child of v.*

Suppose $p = o(1)$ but $p = \omega/n$, so that for any fixed k such that $k^2 p = o(1)$. For any vertex v, the probability of v having exactly $m \leq k$ children has the limiting Poisson distribution with parameter $\lambda = 1$. That is, denote by $c(v)$ the number of children of v then

$$\Pr(c(v) = m) = \frac{e^{-1}}{m!}(1+o(1))$$

To prove the theorem, we examine the probability that the vertex v has exactly $1, 2, \ldots, k$ number of left children.

5.7.1.1 For p is positive and small.

The following approximation is important for the rest of our calculations:

$$\ln(1-p) = -p - \frac{p^2}{2} - \frac{p^3}{3} - \frac{p^4}{4} - \cdots = -p - \frac{p^2}{2}[1 + \frac{2p}{3} + \frac{2p^2}{4} + \ldots]$$

Since $1 \leq 1+\frac{2p}{3}+\frac{2p^2}{4}+\cdots \leq \frac{1}{1-p}$, we have:

$$-p-\frac{p^2}{2}\frac{1}{1-p} = -p(1+\frac{p}{2-2p}) \leq \ln(1-p) \leq -p-\frac{p^2}{2} = -p(1+\frac{p}{2})$$
$$-p(1+O(p)) \leq \ln(1-p) \leq -p(1+O(p))$$

e.g. if $p \leq 1/2$, then: $-p-p^2 \leq \ln(1-p) \leq -p-p^2/2$. Thus, for small and positive p, we take the approximation that $\ln(1-p) = -p(1+O(p))$.

5.7.1.2 The Left Component.

Denote by P_k the probability that there are exactly k *children* to the *left* of v. Then, for $k=0$:

$$\begin{aligned} P_0 &= \prod_{\ell=1}^{\infty}(1-p(1-p)^{\ell-1}) = \exp\Big[\ln\big(\prod_{\ell=1}^{\infty}(1-p(1-p)^{\ell-1})\big)\Big] \\ &= \exp\sum_{j=0}^{\infty}\Big(-p(1-p)^j(1+O(p(1-p)^j))\Big) \\ &= \exp\Big(\sum_{j=0}^{\infty}-p(1-p)^j - O(1)p^2\sum_{j=0}^{\infty}(1-p)^{2j}\Big) \\ &= e^{-1}e^{-O(1)-\frac{p}{2-p}} = e^{-1}(1-O(p)) \end{aligned} \tag{5.15}$$

using $1-x \leq e^{-x} = 1-x+x^2/2-x^3/6+\cdots \leq 1-x+x^2/2$ for $0 \leq x \leq 1$. Similarly, for $k=1$, we get:

$$\begin{aligned} P_1 &= \sum_{j=1}^{\infty}(1-p)^{j-1}p\prod_{\ell\neq j}^{\infty}(1-p(1-p)^{\ell-1}) \approx P_0 p\sum_{j=1}^{\infty}(1-p)^{j-1} \\ &= P_0 p\frac{1}{1-(1-p)} = P_0 \end{aligned}$$

5.7.1.3 Extending to $k=2$.

For $k=2$, we can write P_2 as follows

$$P_2 = \sum_{a_1=1}^{\infty}(1-p)^{a_1-1}p\sum_{a_2=a_1+1}^{\infty}(1-p)^{a_2-1}p\prod_{\substack{\ell=1\\ \ell\neq a_1,a_2}}^{\infty}(1-p(1-p)^{\ell-1}) \tag{5.16}$$

The product in (5.16) can be written as follows

$$\prod_{\substack{\ell=1\\ \ell\neq a_1,a_2}}^{\infty}(1-p(1-p)^{\ell-1}) = \frac{\prod_{\ell=1}^{\infty}(1-p(1-p)^{\ell-1})}{(1-p(1-p)^{a_1})(1-p(q-p)^{a_2})}.$$

It would be useful to generalise the quantity to k variables in the denominator. Particularly, using the series $1/(1-x) = 1+x+x^2+\dots$ we can write $1/(1-p(1-p)^a) = 1+O(p)$. And hence in general case with k, we get

$$\frac{1}{1-p(1-p)^{a_1}}\frac{1}{1-p(1-p)^{a_2}}\cdots\frac{1}{1-p(1-p)^{a_k}} = 1+O(kp) \tag{5.17}$$

For $k=2$ this yields

$$\prod_{\substack{\ell=1\\ \ell\neq a_1,a_2}}^{\infty}(1-p(1-p)^{\ell-1}) = e^{-1}\frac{1-O(p)}{1+O(2p)} = e^{-1}(1+O(p))$$

Thus

$$P_2 = e^{-1}(1+O(p))p^2\sum_{a_1}\sum_{a_2}(1-p)^{a_1-1}(1-p)^{a_2-1}$$

Consider the variables $(1-p)^1,(1-p)^2\dots$ as $x_1,x_2\dots$. The summation is then contributed by the sum of product of two variables x_ix_j

$$\sum_{x_1}\sum_{x_2} = x_1x_2+x_1x_3+\dots+x_2x_3+x_2x_4+\dots \quad \dots$$

Then from the multinomial expansion, we have

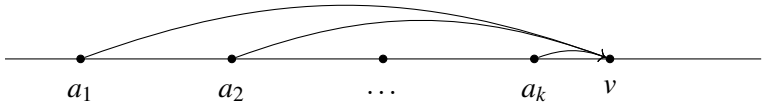
$$\sum_{x_1}\sum_{x_2} = \frac{(x_1+x_2+\dots)^2-(x_1^2+x_2^2+\dots)}{2} \tag{5.18}$$

Substitue in

$$\begin{aligned}\sum_{a_1}\sum_{a_2} &= \frac{(\sum_{\ell=0}^{\infty}(1-p)^{\ell})^2-\sum_{\ell=0}^{\infty}(1-p)^{2\ell}}{2}\\ &= \frac{1}{2}\Big(\frac{1}{p^2}-\frac{1}{1-(1-p)^2}\Big)\end{aligned}$$

Substitue in (5.16), we get

$$\begin{aligned}P_2 &= \frac{p^2}{2}\Big(\frac{1}{p^2}-\frac{1}{1-(1-p)^2}\Big)e^{-1}(1+O(p)) = \frac{1}{2}\{\frac{2-2p}{2-p}\}e^{-1}(1+O(p))\\ &= \frac{1}{2}\Big(1-\frac{p}{2}-\frac{p^2}{2^2}-\frac{p^3}{2^3}+\dots\Big)e^{-1}(1+O(p)) = \frac{1}{2}e^{-1}(1+O(p))\end{aligned} \tag{5.19}$$



5.7.1.4 Extending to fixed k.

Writing down the sum for k

$$P_k = \sum_{a_1=1}^{\infty} p(1-p)^{a_1-1} \sum_{a_2\neq a_1}^{\infty} \dots \sum_{a_k\neq a_1,a_2,\dots}^{\infty} p(1-p)^{a_k-1} \prod_{\ell\neq a_1,\dots,a_k}^{\infty} (1-p(1-p)^{\ell-1}) \quad (a_1<a_2<\cdots<a_k) \tag{5.20}$$

Following the method with P_2, the product in (5.20) can be written as:

$$\prod_{\ell\neq a_1,\dots,a_k}^{\infty} (1-p(1-p)^{\ell-1}) = \frac{\prod_{\ell=1}^{\infty}(1-p(1-p)^{\ell-1})}{\prod_{a_i=a_1,\dots,a_k}(1-p(1-p)^{a_i-1})} = e^{-1}\frac{1-O(p)}{1+O(kp)} \tag{5.21}$$

Since $a_1<a_2<\cdots<a_k$, and applying (5.17):

$$\prod_{a_i=a_1,\dots,a_k} \frac{1}{(1-p(1-p)^{a_i-1})} = 1+O(kp)$$

Thus:

$$\prod_{\ell\neq a_1,\dots,a_k}^{\infty} (1-p(1-p)^{\ell-1}) = e^{-1}(1+O(kp)) \tag{5.22}$$

Now, for the sum in (5.20) with $a_1<a_2<\cdots<a_k$:

$$\sum_{a_1=1}^{\infty} p(1-p)^{a_1-1} \sum_{a_2\neq a_1}^{\infty} \dots \sum_{a_k\neq a_1,a_2,\dots}^{\infty} p(1-p)^{a_k-1} = p^k \sum_{a_1}\dots\sum_{a_k}(1-p)^{a_1-1}\dots$$

Consider the variables in the summation i.e. $(1-p)^{a_1-1}$ as x_1, each term of the sum we are interested is a product of k variables i.e. $x_1x_2\dots x_k$ of order 1, thus by the multinomial theorem its coefficient is $\binom{k}{1!1!\dots} = k!$. Furthermore, from the multinomial theorem, we can write the following bound for the summand:

$$\begin{aligned}(x_1+x_2+\dots)^k - \binom{k}{2}(x_1+x_2+\dots)^{k-2}(x_1^2+x_2^2+\dots) \\ \leq \quad k!\sum_{x_1}\sum_{x_2}\dots\sum_{x_k} \quad \leq (x_1+x_2+\dots)^k\end{aligned} \tag{5.23}$$

The lower bound is taken from the fact that the polynomial $(x_1+\dots)^k$ contains the summand that we are bounding. Whereas the quantity we subtract contains every term in the right hand side, with at

least one repeated term. We show that

$$\Big\{(x_1+x_2+\dots)^k - k!\sum_{x_1}\dots\sum_{x_k}\Big\} \le \binom{k}{2}(x_1+\dots)^{k-2}(x_1^2+\dots)$$

This is indeed true since any variable exists in the left sum, also exists in the right sum. Furthermore, the coefficient of any term in the left hand side is less or equal to the coefficient of any term in the right hand side. For instances, the largest coefficient $k!$ is subtracted. The second largest coefficient is $\binom{k}{2!1!1!\dots} = \binom{k}{2}$ which comes from the terms that have of one variable of order 2, while the rest is of order 1 i.e. $x_1^2 x_2 \dots x_k$. The coefficients of other terms are clearly less than $\binom{k}{2}$, yielding the inequality. The equal case occurs with $k = 2$, as seen in (5.18).

To complete the bound from (5.23), for the upper bound, we have:

$$(x_1+x_2+\dots)^k = \Big(\sum_{\ell=0}^{\infty}(1-p)^\ell\Big)^k = \frac{1}{p^k}$$

Similarly, for the lower bound:

$$(x_1+x_2+\dots)^k - \binom{k}{2}(x_1+x_2+\dots)^{k-2}(x_1^2+x_2^2+\dots) = \frac{1}{p^k} - \binom{k}{2}\frac{1}{p^{k-2}}\frac{1}{1-(1-p)^2}$$

Thus, substitute into (5.23), we get:

$$\frac{1}{p^k} - \binom{k}{2}\frac{1}{p^{k-2}}\frac{1}{1-(1-p)^2} \le k!\sum_{x_1}\sum_{x_2}\dots\sum_{x_k} \le \frac{1}{p^k}$$

Multiply by p^k and e^{-1} and divide by $k!$:

$$\frac{1}{k!}\Big(1 - \binom{k}{2}\frac{p^2}{1-(1-p)^2}\Big)e^{-1} \le P_k \le \frac{1}{k!}e^{-1}$$

The inner term in the the lower bound has been encountered in (5.19), which can be rewritten as $\frac{1}{k!}\Big(1-\binom{k}{2}\frac{p}{2-p}\Big)e^{-1}$, the function of p has the series expansion $p/(2-p) = p/2+p^2/4+p^3/8+\dots = O(p)$. Therefore, for any $k^2p \to 0$:

$$\frac{e^{-1}}{k!}(1-O(k^2p)) \le P_k \le \frac{e^{-1}}{k!}(1+O(kp)) \tag{5.24}$$

□

5.7.1.5 Remark.

In the previous section, we proved that for $p = o(1)$ but $p = \omega(1/n)$, the limiting distribution of left children was Po(1) for any number of left children k satisfying $k^2p = o(1)$. Thus typically we will choose $\lambda = 1$. However the model is completely general and we do not exclude other values of $\lambda \ge 0$.

In general we may try to relate the parameter $\lambda = \lambda(p)$ to p, by generalising equation (5.15):

$$\lambda = -\sum_{\ell=1}^{\infty} \ln(1-pq^{\ell-1}) = \sum_{\ell=1}^{\infty} \ln(\frac{1}{1-pq^{\ell-1}})$$

where $q = 1-p$. When $p = O(\varepsilon)$ we have proved $\lambda \to 1$ where $\varepsilon \to 0$ i.e. $\lambda = 1+O(\varepsilon)$ and if $p = 1-\varepsilon$ we can also show that

$$\lambda = (1+O(\varepsilon))\ln(1/(1-p)) = (1+O(\varepsilon))\ln(1/\varepsilon).$$

For $\varepsilon \leq p \leq 1-\varepsilon$, the evaluation of the above sum is more problematic, but every value of $\lambda \geq 1$ is obtained for some p.

5.7.2 Limiting distribution of left children

In this section, we consider the Poisson limiting distribution for the number of left children of a vertex,

$$P_k = \frac{\lambda^k}{k!}e^{-\lambda}$$

with parameter $\lambda = 1$, for any number k satisfying $k^2 p = o(1)$ to calculate some basic properties of the left subtree rooted at a given vertex v.

5.7.2.1 The distribution of tree sizes

Consider the permutation subgraph on the integers as follows. Suppose each vertex v has a sliding window of length N *vertices* to its left. Note that *we replace n with N* for the number of vertices of the sliding window to avoid conflict of notation in the following sections. Within the window, each edge uv $(u < v)$ exists with probability p i.e. u inserts the front edge uv with probability p. In such case, u is regarded as a direct child of v.

Suppose $p = o(1)$ but $p = \omega/N$. For any vertex v, the probability of v having exactly $m \leq k$ direct children, for any fixed k such that $k^2 p = o(1)$, has limiting Poisson distribution with parameter $\lambda = 1$ i.e.

$$Pr\{c(v) = m\} \sim \frac{e^{-1}}{m!}$$

where $c(v)$ denote the number of children of v.

Theorem 6. *Consider the limiting distribution. Select a vertex v, let T be the tree rooted at v which consists of all left descendants of v. Then the size of the left subtree is given by a power law distribution*

$$Pr(|T| = m) \sim \frac{1}{\sqrt{2\pi}}\frac{1}{m^{3/2}} \tag{5.25}$$

5.7.3 The Lagrange Inversion Formula

The following theorem, known as the Lagrange Inversion Formula is important for the proof below. The theorem statement below is taken from Wilf [63].

Theorem 7 (Lagrange Inversion Formula (LIF))**.** *Let $f(u)$ and $\phi(u)$ be formal power series in u, with $\phi(0) = 1$. Then there is a unique formal power series $u = u(t)$ that satisfies*

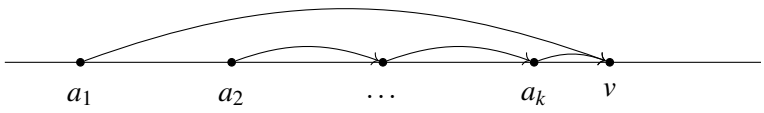
$$u = t\phi(u) \tag{5.26}$$

Further, the value $f(u(t))$ of f at that root $u = u(t)$, when expanded in a power series in t about $t = 0$, satisfies

$$[t^n]\{f(u(t))\} = \frac{1}{n}[u^{n-1}]\{f'(u)\phi(u)^n\}$$

where the $[t^n]\{f(u(t))\}$ denote the n^{th} coefficient of the series $\{f(u(t))\}$.

5.7.4 Proof of Theorem 6



Consider a tree T rooted at v. By Theorem 5, the LST is *empty* with probability e^{-1}, in which case we say T is an empty root. Otherwise, for any vertex u that is a direct child of v, u is also a root and its number of children is also given by Theorem 5. Thus, *traversing* leftward down the tree, we can calculate the size of tree T by recursively examining each child, as long as there remains any.

Let $F(x)$ be the probability generating function for the size of the left subtree (LST) of the tree T rooted at v. Now T is either an empty root; or a root that has exactly k children (probability P_k), which are also roots which have the same distribution for the number of children. Thus:

$$\begin{aligned} F(x) &= x(e^{-1} + e^{-1}F + \frac{e^{-1}}{2!}F^2 + \frac{e^{-1}}{3!}F^3 + \ldots) \\ &= xe^{-1}(1 + F + \frac{F^2}{2!} + \frac{F^3}{3!} + \ldots) = xe^{-1}e^F \end{aligned}$$

The above functional equation is in the form of (5.26), which can be solved by applying the Lagrange Inversion Formula. First, rewrite it as

$$eF = xe^F$$

let $w(x) = eF(x)$, thus

$$w(x) = xe^{w(x)/e} = x(e^{1/e})^{w(x)} = xc^{w(x)}$$

by putting $e^{1/e} = c$. Next, to apply the LIF, let $f(u) = u$ with $\phi(u) = c^u$, note that $\phi(0) = 1$ thus the LIF applies. For any $k > 0$ we get

$$[x^k]w(x) = \frac{1}{k}[u^{k-1}]\{\phi^k(u)\} = \frac{1}{k}[u^{k-1}]\{c^{ku}\} \tag{5.27}$$

For the power series $\{c^{ku}\}$, let $ku = y$, we have:

$$c^y = \sum_{j\geq 0} \frac{\ln^j(c)}{j!} y^j = \sum_{j\geq 0} \frac{k^j \ln^j(c)}{j!} u^j$$

Thus, the k^{th} coefficient of the series $w(x)$ is obtained by extracting the $(k-1)^{th}$ coefficient of the series $\{c^{ku}\} = \sum_j \frac{k^j \ln^j(c)}{j!} u^j$, as stated in (5.27). Hence

$$[x^k]w(x) = \frac{1}{k}\frac{k^{k-1}\ln^{k-1}(c)}{(k-1)!} = \frac{k^{k-1}\ln^{k-1}(e^{1/e})}{k!} = \frac{k^{k-1}}{k!e^{k-1}}$$

The Stirling's approximation for $k!$ is

$$\sqrt{2\pi k}(k/e)^k \leq k! \leq \sqrt{2\pi k}(k/e)^k e^{1/12k}$$

thus $k! = \sqrt{2\pi k}(k/e)^k e^{O(1/k)}$. Finally, we get:

$$[x^k]w(x) = \frac{k^{k-1}}{\sqrt{2\pi k}(k/e)^k e^{O(1/k)} e^{k-1}} = \frac{1}{\sqrt{2\pi}}\frac{e^{1-O(1/k)}}{k^{3/2}}$$

Finally to extract the coefficient of $F(x)$, recall that we have put $w(x) = eF(x)$, therefore $[x^k]F(x) = [x^k]w(x)/e$, hence

$$[x^k]F(x) = (1 - O(1/k))\frac{1}{\sqrt{2\pi}}\frac{1}{k^{3/2}} \tag{5.28}$$

□

5.7.5 Remark

Fix a vertex u, consider a vertex v $(u < v)$ such that there are $k-1$ vertices between u and v; for $1 \leq k \leq n$ so that u is *inside the sliding window* of v. The probability of the event R that u connects to v is

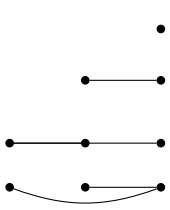
$$Pr(R) = p(1-p)^{k-1}$$

this occurs because u disconnects from every vertex prior to v and finally attaches to v. Thus the probability that u does not connect to v is $1 - Pr(R)$.

Therefore, the probability that u attaches to *no one to its right* is

$$\prod_{k=1}^{n}(1-p(1-p)^{k-1}) = e^{-1+o(1)}$$

similar to equation (5.14). Thus, the probability that u is a singleton (or disconnected from every one) is asymptotically:



$$Pr = e^{-1}e^{-1}$$

$$Pr = e^{-1}e^{-1}$$

$$Pr = \frac{1}{2}e^{-1}e^{-1}$$

Thus the distribution of a complete tree of size up to k is then

$$P(k) = \frac{1}{k!}e^{-2} \tag{5.29}$$

5.7.5.1 The expected left component size inside the window.

Informally, in the "infinite" model the probability that the entire left subtree of size m fits in a window of size N is at most (5.28). So in the sliding window model, for m constant or tending slowly to ∞.

$$Pr(\text{LST} \geq m) \leq (1-O(1/m))\frac{1}{\sqrt{2\pi}}\int_{m}^{\infty} m^{-3/2}dm = (1+o(1))\frac{1}{\sqrt{m}}\sqrt{\frac{2}{\pi}}$$

Using (5.28), we get:

$$E[\text{component size}] \leq \frac{1}{\sqrt{2\pi}}\sum_{m\geq 1}^{N}\frac{m}{m^{3/2}} = \frac{1}{\sqrt{2\pi}}\sum_{m\geq 1}^{N}\frac{1}{m^{1/2}} = H_{N,1/2}$$

in which $H_{N,1/2}$ is the generalised Harmonic number. As the function is decreasing, we can bound it from Riemann sums:

$$\int_{1}^{N} m^{-1/2}dm < \sum_{1}^{N} m^{-1/2} < \int_{0}^{N-1} m^{-1/2}dm$$

$$2\sqrt{N}-2 < \sum_{1}^{N} m^{-1/2} < 2\sqrt{N-1}$$

And therefore the expected size of the LST is $O(\sqrt{N})$.

5.8 Conclusion

In this chapter, we studied, in more detail, various properties of the *permutation subgraph* of a Erdos-Reyni random graph $G(n,p)$. We showed that, for some specific ranges of the variable p, a permutation subgraph of a random graph: has a giant component, the location of the roots. Further, we computed some properties of the subgraph, including: the expected number of components, the probability that the subgraph is connected and the expected path length. In the remaining sections, we extend the random graphs to the integers where each vertex v has a sliding window of length n to its left in which vertices can attach itself to v with some probability. We showed that the number of children of every vertex (up to some specific value) has the limiting Poisson distribution with $\lambda = 1$.

Chapter 6

Analysis of the burn-edge fragmentation algorithm

6.1 Definition of the burn-edge process

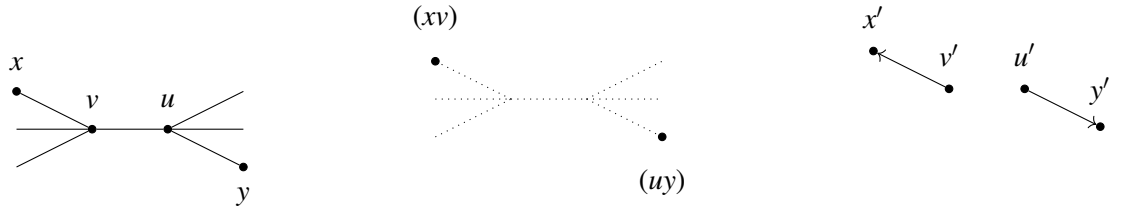
In previous sections, we studied some *mapping processes* where mappings are made between vertices. In this chapter, we introduce a process which performs mapping on *edges*.

Suppose the graph $G(V,E)$ is a simple, undirected graph. Let $G'(V',E')$ be an empty graph, we copy the vertex set of G to G' i.e. $V' \leftarrow V$ and $E' \leftarrow \emptyset$. Objects in G is denoted *without a prime* symbol i.e. v; objects in G' is denoted with a *prime* symbol v'. Generally, in each iteration, the following steps are done, in order, until options are exhausted.

1. Select an edge $e(v,u)$ in G,
2. Let x and y be the adjacent vertices of v and u, respectively, chosen typically at random. Then x absorbs v and y absorbs u. This means, we do, for example for u,: remove u from the set of vertex $V = V \setminus \{u\}$; along with its incident edges: $\forall w \in Adj(u) : \quad Adj(w) = Adj(w) \setminus \{u\}$. Similarly for v.
3. v and u become *inactive*.

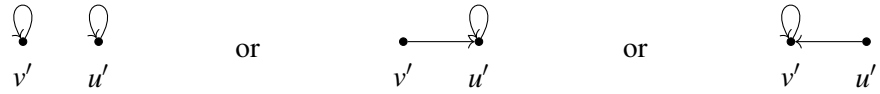
At the end, *all remaining vertices have degree* 0 *are roots vertices*. In more detail, in step one an edge $e(v,u)$ is selected, typically at random from the set of edges. Next, v and u are *mapped to adjacent neighbours* i.e. x and y, respectively. The *neighbours* x and y are chosen according to some rules which are discussed in more detail in following sections. We then remove the edge $e(v,u)$ along with all edges incident with vertices v and u. Alternatively, we can consider that *simultaneously*, x and y *absorb* v and u, respectively. In G' we direct two edges: (v',x') and (u',y'). The absorbed vertices become *inactive* and unavailable for future computation. We call this operation *burning* the edge $e(v,u)$.

We provide an illustration of the process as follows. Suppose we have selected an edge $e(v,u)$, and there are two adjacent vertices x, y. The operation: *burning* the *edge* $e(v,u)$ is equivalent to coalescing the pairs v,x and u,y to simple vertices. All edges incident with $\{v,u\}$ are removed. Hence v and u are considered *inactive* and are not available for future computations. The directed edges $e(v',x')$ and $e(u',y')$ are added to the graph G'. These edges are naturally directed, but in some cases we can choose to ignore its orientation.

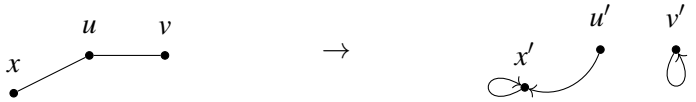


(a) Input G (b) Output G (c) Output G'

It does not matter to the future of the algorithm which neighbours are chosen. It does, however, affect the number of components. For example, consider a path of length 2 i.e there is a single edge $e(v,u)$ then there are three possible cases can occur, which produce different numbers of components



Furthermore, if the two latter cases are allowed, then an orientation like $u \to v \to y$ could also occur at any iteration. In this study, we do not consider these situations. In other words, we do *not allow the vertices of the selected edge to map to each other*. More specifically, for a selected edge $e(v,u)$, if $d(v) = 1$ then v absorbs itself and becomes *root*; the same applies to u. There would be situations in which there is a path of length 3, for which the following decision is made. Assume the selected edge is $e(u,v)$



At the end of the algorithm, the result G' is a set of tree-like structures, each with a unique *root*. Hence, the number of root vertices is also the number of components.

In the following section, we analyse the edge-burning algorithm on the cycle graph. We then introduce a variation of the algorithm and analyse it on 2-circulant graph. For illustrations of the algorithms, we skip the copy graph G' as seen above. Instead, we just draw the directed edges directly on the graph G.

The analysis in this chapter are structured similarly to those in Chapter 3, as described in Section 3.3, page 22.

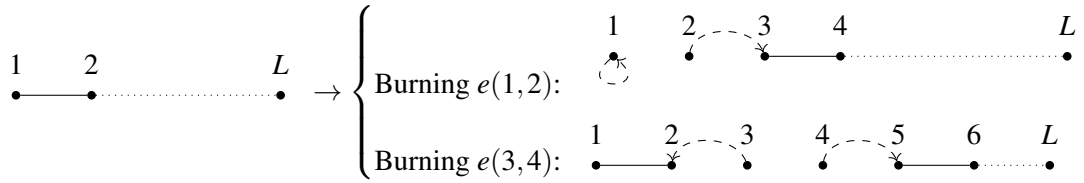
6.2 Analysis of burn-edge process on cycle graph

Let the input be a cycle C. In the first iteration, a random edge is selected and *burned*. Consequently, C is reduced to a path P. Therefore, the *number of components* of a *burn-edge* on a cycle C is given by the same process on a path P with two less vertices. We use an unconventional notation: denote by L the *length of the path* which is *the number of vertices*. It follows that for a path of length L, the number of edges is $L-1$. Denote by N_L the expected number of components produced by applying the algorithm to a path of length L.

Let the set $V = \{1,2,\ldots,L\}$ be the vertex set of a path P of length L. Then the following are the unique cases for *burning* the edges in P:

- $e(1,2)$ is burned i.e. the incident edges of the end points;
- $e(i,i+1),(2 \leq i \leq L-2)$ is burned i.e. the *inner* edges.

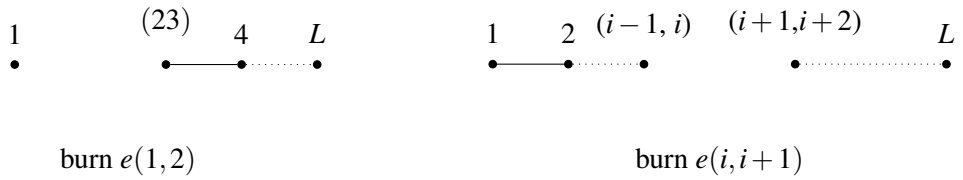
Note that if $L = 1$ (a single vertex) then that vertex is a root. For each case listed above, the result is that the initial path is broken into two paths of shorter length, as illustrated below (the mapping edges are drawn in *dashed* lines)



In the *first case*, the edge incident to either of end-points i.e. $e(1,2)$ (or $e(L-1,L)$) is burned. Then as we forbid the pair of vertices to map to each other, the former maps to itself and becomes a root; while the latter maps to vertex 3. Another way to think about this is simultaneously vertex 1 absorbs itself; and vertex 3 absorbs vertex 2. The results consists of an isolated vertex and a path of length $L-3$.

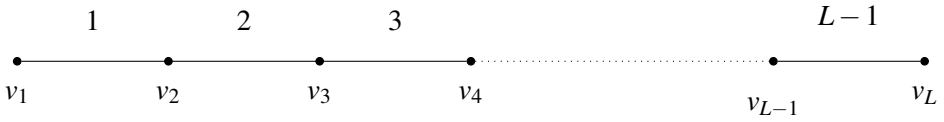
In the *second case*, an *inner* edge is selected which connects the pair of vertices $(i,i+1)$ for $2 \leq i \leq L-2$ is burned. Simultaneously, vertex $i-1$ absorbs vertex i and $i+2$ absorbs vertex to $i+1$. The results are two shorter path of length $i-1$ and $L-i$.

Note that, the absorbed and absorbing are treated as *a single vertex* in future calculations. This means that we can estimate the expected number of components from a path of length L. However, we are not able to obtain the distribution of component sizes using this method.



6.2.1 Recurrence formulation

Let the input be a path P of length L with vertex set $V = \{v_1, v_2, \ldots, v_L\}$. Furthermore, let the edge set be $E = \{e_1, e_2, \ldots, e_{L-1}\}$ where an edge with index i $(1 \leq i \leq L-1)$ connects the pair of vertices (v_i, v_{i+1}).



It can be verified that if $L = 1$, then $N_L = 1$. If $L = 2$ the path is a single edge, then $N_2 = 2$ because both vertices become roots, according to the rule. Assuming for $s > j$ the sum $\sum_{i=s}^{j} N(i) = 0$, we have the following

$$N_L = \sum_{1 \leq i \leq L-1} N(i) \qquad (L \geq 3, N_0 = 0, N_1 = 1, N_2 = 2)$$

in which i is the index of the *burned* edge. Hence

$$N(i) = \begin{cases} 1/(L-1)(N_1 + N_{L-2}) & (i = 1, L-1) \\ 1/(L-1)(N_{i-1} + N_{L-i-1}) & (2 \leq i \leq L-2) \end{cases}$$

Multiplying N_L by $L-1$ and expanding, we get

$$(L-1)N_L = 4N_1 + 2N_{L-2} + 2N_{L-3} + \sum_{i=3}^{L-3} N_{i-1} + \sum_{i=3}^{L-3} N_{L-i-1}$$

Since the two sums are interchangeable, redistributing the remaining terms into the sum yields

$$(L-1)N_L = 2N_1 + 2\sum_{i=1}^{L-2} N_i$$

Similarly for N_{L-1}

$$(L-2)N_{L-1} = 2N_1 + 2\sum_{j=1}^{L-3} N_j$$

Subtracting $(L-2)N_{L-1}$ from $(L-1)N_L$, we have the following recurrence

$$(L-1)N_L = (L-2)N_{L-1} + 2N_{L-2} \quad (L \geq 3, N_0 = 0, N_1 = 1, N_2 = 2) \tag{6.1}$$

6.2.1.1 Generating function formulation

Replace N_L by a_n the recurrence from equation (6.1) can be rewritten as

$$(n-1)a_n = (n-2)a_{n-1} + 2a_{n-2} \quad (n \geq 3, a_0 = 0, a_1 = 1, a_2 = 2)$$

Let $G(x) = \sum_n a_n x^n$ be the generating function of the sequence of a_n. Multiply both side by x^n and sum on $n \geq 3$ yields:

$$\sum_{n\geq 3}(n-1)a_n x^n = \sum_{n\geq 3}(n-2)a_{n-1}x^n + 2\sum_{n\geq 3} a_{n-2}x^n \tag{6.2}$$

The terms in (6.2) can be rewritten according to $G(x)$ as follows. Note that the summations above are taken for $n \geq 3$, hence we must subtract the terms with $n \leq 2$.

- $\sum_{n\geq 3}(n-1)a_n x^n = \sum_{n\geq 3} n a_n x^n - \sum_{n\geq 3} a_n x^n = \big(xG'(x) - x - 4x^2\big) - \big(G(x) - x - 2x^2\big)$
- $\sum_{n\geq 3}(n-2)a_{n-1}x^n = \sum_{n\geq 3} n a_{n-1}x^n - 2\sum_{n\geq 3} a_{n-1}x^n = \big(x^2G'(x) + xG(x) - 2x^2\big) - 2\big(xG(x) - x^2\big)$
- $\sum_{n\geq 3} a_{n-2}x^n = 2x^2G(x)$

Putting everything together, we can rewritte (6.2) as

$$xG'(x) - x - 4x^2 - (G(x) - x - 2x^2) = x^2G'(x) + xG(x) - 2x^2 - 2(xG(x) - x^2) + 2x^2G(x)$$

Rearranging terms yields the following first order differential equation

$$G'(x) + G(x)\frac{2x^2 - x + 1}{x^2 - x} = \frac{2x}{1-x} \tag{6.3}$$

Following the standard routine for solving such differential equations (see Section 12.6 for more details), we first find the integrating factor $\mu(x)$ which is given by

$$\mu(x) = e^{\int (2x^2 - x + 1)/(x^2 - x)} = e^{2x + 2ln(x-1) - ln(x)} = (x-1)^2 e^{2x}/x$$

Multiply both sides in (6.3) with the integrating factor then take the indefinite integral of both sides

$$G(x)(x-1)^2e^{2x}/x = 2\int (1-x)e^{2x}dx + C$$

The integrand gives $e^{2x}(3-2x)/4$, thus $G(x)$ is

$$G(x) = \frac{x(3-2x)}{2(x-1)^2} + C\frac{x}{(x-1)^2e^{2x}}$$

Now, we have to solve for C, since $G(0) = 0$ for any value of C. We have to solve for $G'(0) = a_1 = 1$. We have

$$G'(x) = \frac{x-3}{2(x-1)^3} - C\frac{2x^2-x+1}{e^{2x}(x-1)^3}$$

Hence $G'(0) = 3/2 + C = 1$ thus $C = -1/2$. And finally

$$G(x) = \frac{x(3-2x)}{2(x-1)^2} - \frac{1}{2}\frac{x}{(x-1)^2e^{2x}} = \frac{x}{1-x} + \frac{1}{2}\frac{x}{(x-1)^2} - \frac{1}{2}\frac{x}{(x-1)^2e^{2x}} \tag{6.4}$$

6.2.1.2 Asymptotic of the coefficients $[x^n]G(x)$

We are interested in the number of expected components N_L which is given by the n^{th} coefficient $[x^n]G(x)$. It is possible to expand this series. However it is problematic to obtain an exact closed form of the coefficients because of the last function. Hence we look at the asymptotic growth of $[x^n]$.

The first function is a geometric series, of which the coefficients are 1. The second function is also an elementary series, its n^{th} coefficient is $[x^n]\frac{x}{(x-1)^2} = n$. Let the last function be $f(x)$. It has a single pole of order 2 at $x = 1$, then the coefficients $[x^n]f(x)$ are well approximated by the coefficients of its principal parts of its Laurent expansion at $x = 1$ (see Section A.2.1 for more detail). We have

$$\begin{aligned} f(x) = -\frac{1}{2}\frac{x}{(x-1)^2e^{2x}} &= -\frac{1}{2}\Big(\frac{1}{x-1} + \frac{1}{(x-1)^2}\Big)\frac{e^{-2(x-1)}}{e^2} \\ &= -\frac{1}{2e^2}\Big(\frac{1}{x-1} + \frac{1}{(x-1)^2}\Big)\Big(1 - 2(x-1) + \frac{4(x-1)^2}{2!} - \frac{8(x-1)^3}{3!} + \dots\Big) \\ &= -\frac{1}{2e^2}\Big(\frac{1}{(x-1)^2} - \frac{1}{x-1} + \frac{2}{3}(x-1) - \dots\Big) \end{aligned}$$

Hence

$$[x^n]f(x) \sim [x^n]\Big\{-\frac{1}{2e^2}\Big(\frac{1}{(x-1)^2} + \frac{1}{1-x}\Big)\Big\} = -\frac{1}{2e^2}(n+2)$$

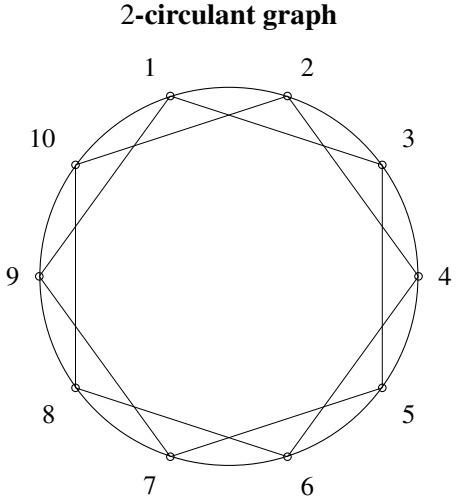
Putting everything together we get

$$[x^n]G(x) \sim 1 + \frac{1}{2}n - \frac{1}{2e^2}(n+2) \approx (n+2)(1/2 - 1/(2e^2)) \approx (n+2)\times 0.4323 \qquad \square$$

6.3 Modified burn-edge on circulant graph

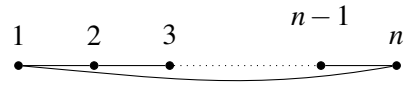
In this section, we study a variation of the burn-edge process and extend its analysis on circulant graphs. A k-circulant graph C_n^k is a graph in which there are n vertices $V = \{0,1,2,\ldots,n-1\}$ such that the i^{th} vertex is adjacent to the $i \pm j \mod n$ for each $j \in [1,k]$. Thus, the 1-circulant graph is the cycle C_n i.e. vertex i is connected to vertex $i-1$ and $i+1$. The $\lfloor n/2 \rfloor$-circulant graph is the complete graph K_n.

Consider a 2-circulant graph C_n^2. The graph can be constructed by firstly generating a cycle of n vertices. Next we connect every vertex to the vertices at distance 2 from it i.e. the neighbours of the adjacent vertices. Thus, every *odd*-index vertex is now connected two nearest *odd*-index neighbours; and similarly for *even*-index vertices. In other words, the graph can be 'decomposed' into three cycles. Henceforth, we assume that there are an even number of vertices. Also, we change the *indexing of vertices* to $\{1,2\ldots,n\}$ for convenience in future calculation. We give an example for the case $n = 10$.

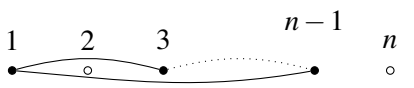


Decomposition of a 2-circulant graph

- *'Main'* cycle:



- *'Odd'* cycle:



- *'Even'* cycle:

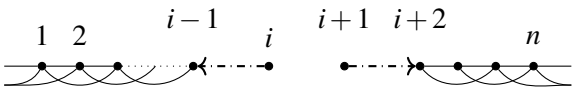
Fig. 6.1 An example of a 2-circulant graph and its decomposition into *main, odd* and *even* cycles.

Definition 1. *A main edge is an edge in the 'main' cycle, connecting the i^{th} vertex where $i = \{1,2,\ldots,n\}$ with the $(i+1)^{th}$, also including the edge $(n,1)$.*

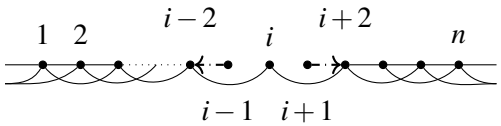
Definition 2. *An odd/even edge is an edge in the odd/even cycle, connecting the i^{th} vertex where $i = \{1,2,\ldots,n\}$ with the $(i+2)^{th}$ for i is odd/even, also including the wrapped around edge e.g. $(n,2)$.*

Consider a *burn-edge* process on C_n^2, in the first iteration there are two unique cases corresponding to the type of edge which is *burned*. The main difference is illustrated as follows:

- Burn *main* i.e. $e(i,i+1)$:



- Burn *even or odd* i.e. $e(i-1,i+1)$:



In the first case, if a *main* edge is burned, then the resulting structure is broken. In other words, imagine the graph is layout in a circle like in Figure 6.1, then it is no longer possible to traverse the graph clockwise. Whereas in the *second case*, the resulting structure is still *connected* with two bridges: $(i-2,i)$ and $(i,i+2)$. Hence in this case, we can still traverse the graph in a clockwise manner i.e. $i-2,i,i+2$.

The effect of burning a *main edge* on a circulant graph is therefore similar to that of burning any edge on a cycle. That is, with each edge burned, the resulting object has the same structure as the input, only with shorter length (with the only exception in the first iteration). On a cycle the first burned edge reduces it to a path. On a circulant graph, the input graph reduces to the structure illustrated in Figure 6.2. Thereafter, for cycles, the recurrent structures are paths. For 2-circulant graph, the object generated is a cycle-like structure. Such that for every vertex $i \in V = \{1,2,\ldots,n\}$, it is adjacent to the vertices $1 < i \pm 1, 2 < n$. We have removed the mod operation (when constructing the circulant graph), thus the resulting structure is no longer connected in a *circulant* manner.

This resulting graph has the degree sequence: $\{4,\ldots,4,3,3,2,2\}$. We denote a graph with this topology as $C_n^{2\oplus}$. A visualisation for $C_n^{2\oplus}$ is provided below.

Definition 3. *The $C_n^{2\oplus}$ graph is a graph with n vertices, generated from a 2-circulant with (n+2) vertices by burning a **main** edge.*

6.3.1 Burning only *main* edges on $C_n^{2\oplus}$

Let the input graph be a $C_n^{2\oplus}$. Consider a *restricted* edge-burning process in which only *main-edges* are burned. Suppose an edge $e(i,i+1)$ in $C_n^{2\oplus}$ $(n \geq 5)$ is *burned*, then vertices i and $(i+1)$ become *inactive*. Consequently, the input graph is broken into two separated substructures. These substructures

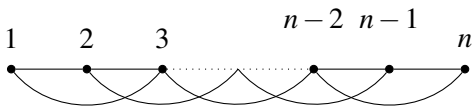


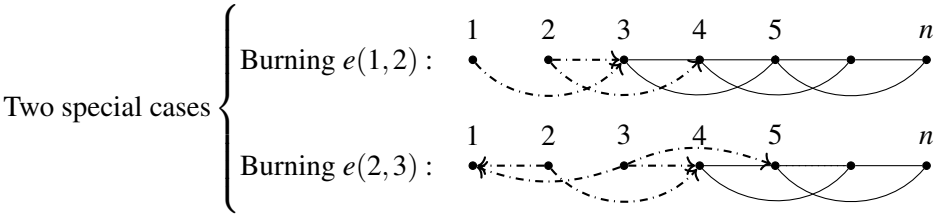
Fig. 6.2 The structure generated by burning a *main edge* from the 2-circulant graph. We denote this structure as $C_n^{2\oplus}$ graph, which has the degree sequence $\{4,\ldots,4,3,3,2,2\}$.

are, $C_{i-1}^{2\oplus}$ and $C_{n-i-1}^{2\oplus}$, thus a recurrence yields. Now we need to examine the two special cases i.e. burning the first (and last) and the *next-to-last* main edges.

If the main edge $e(1,2)$ is burned. Then vertex 1 can only map to vertex 3. Whereas vertex 2 has two options of 3 or 4. In either case, the structure is only *pruned* with *two* vertices going *inactive*. In the end, the resulting structure is the graph $C_{n-2}^{2\oplus}$.

If the main edge $e(2,3)$ is burned. Vertex 2 can be mapped to 1 or 4; and vertex 3 to $1,4,5$. In either case, 1 then becomes a *root*. It follows that the resulting structures are $C_1^{2\oplus}$ and $C_{n-3}^{2\oplus}$.

These special cases are illustrated in the figure below, the *directed, dashed-dotted* line represent all possible choice of the source vertex.



Let the input graph be $C_L^{2\oplus}$ where L is the number of vertices of the graph. Let the vertex set be $V=\{v_1,v_2,\ldots,v_L\}$. Let the set of *main edges* be $E=\{1,2,\ldots,L-1\}$, in which each edge i connects the pair of vertices (v_i,v_{i+1}). Denote by N_L the *expected number* of components of $C_L^{2\oplus}$.

It can be verified that the expected number of components of a $C_1^{2\oplus}$ is $N_1=1$ since the input is an isolated vertex. Further, $C_2^{2\oplus}$ is $N_2=2$ i.e. if the input is a path of length 2 (a single edge) then we have 2 roots. Further, we have $N_3=1$ and $N_4=2$, see Figure 6.3. We have the recurrence

$$N_L=\sum_{1\le i\le L-1} N(i) \qquad (L\ge 4, N_0=0, N_1=1, N_2=2, N_3=1)$$

in which i is the index of the selected main edge and assume for $s>j$ the sum $\sum_{i=s}^{j} N(i)=0$. For each edge i we have

$$N(i)=\begin{cases} 1/(L-1)N_{L-2} & i=1,L-1 \\ 1/(L-1)(N_{i-1}+N_{L-i-1}) & 2\le i\le L-2 \end{cases}$$

Multiply N_L by $L-1$ and expand gives

$$(L-1)N_L = 2N_{L-2} + \sum_{i=2}^{L-2} N_{i-1} + \sum_{j=2}^{L-2} N_{L-j-1} = 2N_{L-2} + 2\sum_{i=1}^{L-3} N_i$$

Carry out similarly for N_{L-1} then subtract $(L-2)N_{L-1}$ from $(L-1)N_L$ we obtain

$$(L-1)N_L = (L-2)N_{L-1} + 2N_{L-2} \tag{6.5}$$

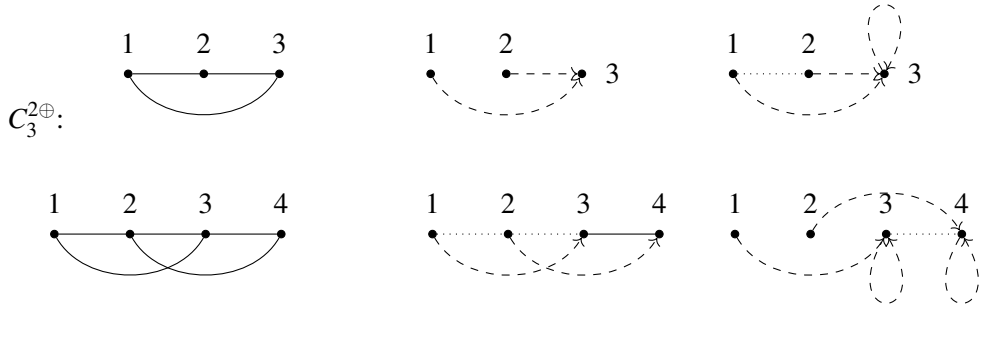


Fig. 6.3 Illustration of the base cases, in the figures the *dotted lines* indicate to-be-removed edges i.e. incident edges of inactive vertices.

If the input graph is $C_3^{2\oplus}$. Suppose the burned edge is $e(1,2)$, then both vertices have only one option: map to 3. Next iteration, only 3 remains thus it becomes a root.

If the input graph is $C_4^{2\oplus}$, suppose the burned edge is $e(1,2)$. Furthermore, assuming that 1 points to 3 and 2 points to 4. Then there remains a single edge $e(3,4)$. Hence 3 and 4 become two roots.

6.3.1.1 Generating function formulation

Substituting N_L with a_n, let the OGF $G(x) = \sum_n a_n x^n$, then multiply by x^n and sum on $n \geq 4$

$$\sum_{n\geq 4} n a_n x^n - \sum_{n\geq 4} a_n x^n = \sum_{n\geq 4} n a_{n-1} x^n - 2\sum_{n\geq 4} a_{n-1} x^n + 2\sum_{n\geq 4} a_{n-2} x^n \tag{6.6}$$

Equation (6.6) is the same as (6.1) except for the initial conditions. Thus, using results from previous section, we can relate the terms in (6.6) to $G(x)$ as follows

$$\Big(xG'(x) - (x+4x^2+3x^3)\Big) - \Big(G(x) - (x+2x^2+3x^3)\Big) = x^2\Big(G'(x) - (1+4x)\Big) - x\Big(G(x) - (x+2x^2)\Big) + 2x^2\Big(G(x) - x\Big)$$

A differential equation is then obtained by rearranging the terms

$$G'(x) + G(x)\frac{2x^2 - x + 1}{x^2 - x} = 2x$$

$G(x)$ here is only slightly different from (6.3), we obtain

$$G(x) = \frac{x(2x^2-6x+5)}{2(x-1)^2} + C\frac{x}{e^{2x}(x-1)^2} \tag{6.7}$$

Using $G'(0) = 1$ gives $C = -3/2$, substituting into (6.7), simplify and rearranging we obtain the final form for $G(x)$

$$G(x) = \frac{x^2}{x-1} + \frac{2x}{1-x} + \frac{1}{2}\frac{x}{(x-1)^2} - \frac{3}{2}\frac{x}{(x-1)^2e^{2x}} \tag{6.8}$$

6.3.1.2 Asymptotic growth of the coefficients $[x^n]G(x)$

The first three terms of $G(x)$ in 6.8 are three elementary OGFs, of which the coefficients $[x^n]$ are $-1, 2, n/2$, respectively. The last term of $G(x)$ is the same as in Equation 6.4 (page 94) in Section 6.2.1.1. Thus, using previous results we get

$$[x^n]G(x) \sim -1+2+\frac{1}{2}(n+1) - \frac{3}{2e^2}(n+2) \approx (n+2)(1/2-3/2e^2)\times 0.2970$$

□

6.3.2 Burning *odd* or *even* edges on $C_L^{2\oplus}$

Consider a modified burn-edge process on $C_L^{2\oplus}$. For this variation, we only select an *odd or even* (odd/even) edge and burn it. Suppose the chosen edge is $e(i-1, i+1)$. We pointed out that by burning this edge, the resulting structure is connected i.e. by two incident edges of vertex i. Thus, we further modify the rule by mapping also vertex i. This ensures the resulting structure is disconnected, and hence it is possible to derive a recurrence to estimate the expected number of components. More specifically, suppose an edge $e(i-1, i+1)$ is selected, to burn such edge, we perform the following operations

1. Burn the edge $e(i-1, i+1)$. Particularly, simultaneously, $i-1$ is absorbed by vertex j for some $j \neq i+1$; and $i+1$ is absorbed by vertex k for some $k \neq i-1$. If there is no valid j or k, vertices $i-1$ or $i+1$ become root.

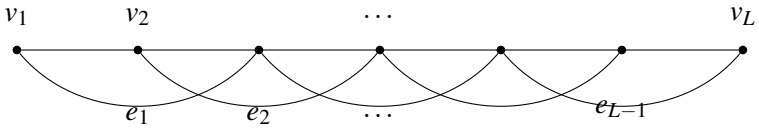
2. If there exists an active vertex i, map vertex i to any active vertex h, such that $h \neq i-1, i+1$. If i has no active neighbour, i is mapped to itself.

3. The three vertices $i-1, i, i+1$ become *inactive*.

The result is the input graph $C_L^{2\oplus}$ is broken into two structures of shorter length, particularly, $C_i^{2\oplus}$ and $C_j^{2\oplus}$ such that $i+j+3 = L$.

Let the input graph be $C_L^{2\oplus}$ with the vertex set $V = \{v_1, v_2 \ldots, v_L\}$. We label the set of odd and even edges using the index of the left vertex i.e. e_1 connects the pair (v_1, v_3), e_2 connects (v_2, v_4) and so on. Consequently, the number of odd/even edges is $L-2$. Furthermore, if L is even then the

number of *odd* edges is equal to of *even* edges; else the number of *even* edge is one less than the number of odd edges. Nevertheless this has no effect on the burn-edge process.



6.3.2.1 Illustration of base cases: for $L = 1,2,3,4$

Let N_L be the expected number of components of the input graph $C_L^{2\oplus}$. Then $N_0 = 0, N_1 = 1$. If $L = 2$ the graph is a single main edge. Since there is no odd/even edge, $N_2 = 1$, i.e. vertex 1 is absorbed by vertex 2 which then becomes a root (we note that this is a special case).

For $L = 3$, vertex 1 and 3 have only one option of mapping to 2 then 2 to itself, therefore $N_3 = 1$. For $L = 4$, there are two available odd/even edges, burning either edge will generate a path of length 2, thus $N_4 = \frac{1}{2}N_2 + \frac{1}{2}N_2 = 1$.

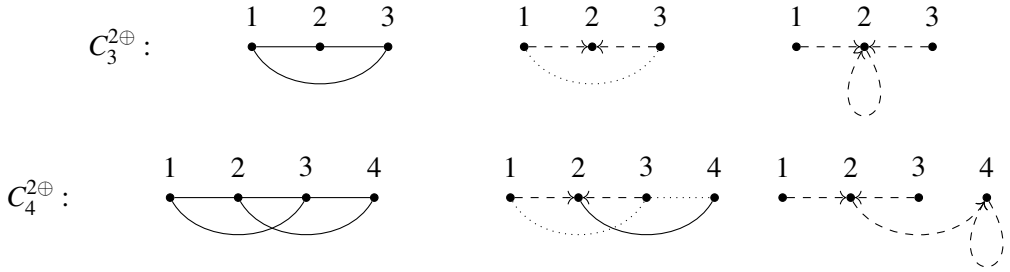


Fig. 6.4 Illustration of burning odd/even edges $C^{2\oplus}$ of length 3 and 4. The *dotted lines* indicate edges to be removed; *dashed, directed line* indicate the mapping and *solid line* indicate edges remain in the graph.

For case $C_3^{2\oplus}$, since vertex 1 and 3 cannot map to each other, they map to 2. Then 2 becomes a root, since 2 is not allowed to map to either vertex of the selected edge.

For case $C_4^{2\oplus}$, assuming the odd edge $e(1,3)$ is selected. Both 1 and 3 maps to 2, then 2 maps to 4 - its only active neighbour.

6.3.2.2 Illustration of the burn-odd/even edge algorithm on the graph $C_5^{2\oplus}$

We provide an example for the algorithm on the graph $C_5^{2\oplus}$ as follows. Note that this case with $L = 5$ vertices is not one of the base cases. This example serves as an illustration of the algorithm. Suppose the edge $e(1,3)$ is burned, the following are the possible selections for vertices $1,3$ and then 2

1. Vertex 1 has only one available neighbour 2, thus 1 is absorbed by 2.

2. Vertex 3 has three available neighbours: $2,4,5$, thus one is selected at random hence one of the events 3 is absorbed by $2,4,5$ occurs with equal probability.

3. Recall that vertex 2 cannot map to 1 and 3, thus the only available neighbour is 4. Thus, 2 is absorbed by 4.

In the end, vertices $1,3$ then 2 are absorbed and become inactive. The remaining vertices are 4 and 5. Hence the structure produced is $C_2^{2\oplus}$ and $N_2 = 1$ (this is a special case that have been mentioned above). Hence the expected number of components for this case is 1.

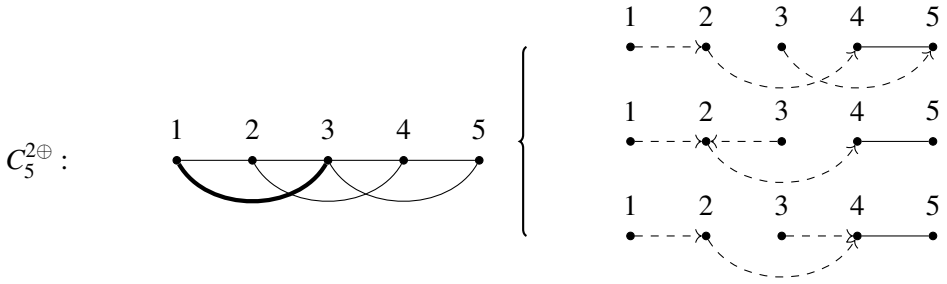


Fig. 6.5 Input graph: $C_5^{2\oplus}$; burning the edge $e(1,3)$. The *dashed* edges represent the *mapping* (absorb) edge. The *solid* edge represent the remaining edges after the iteration.

Now suppose the edge $e(2,4)$ is burned. Then, for each vertex to be absorbed i.e. $2,4$ then 3 there are two available options. For vertex 2 the options are 1, 3. Vertex 4 selects either 3 or 5. Vertex 3 can be absorbed by either 1 or 5. The following table and figure provide illustrations of these cases.

Vertex	Absorbed by							
2 4	1 3	1 5	1 3	1 5	3 3	3 5	3 3	3 5
3	1	1	5	5	1	1	5	5
Figure	(a)	(b)	(c)	(d)	(e)	(f)	(g)	(h)

At the end, vertices 1 and 5 remain with degree 0 thus become roots. Hence, for this case, the expected number of components is 2.

Burning the edge $e(3,5)$ is similar to burning the edges $e(1,3)$, for which the expected number of components is 1. Thus, combining all three cases of burning the edges $e(1,3), e(2,4), e(3,5)$, we get $N_5 = 1/3 + 2/3 + 1/3 = 4/3$.

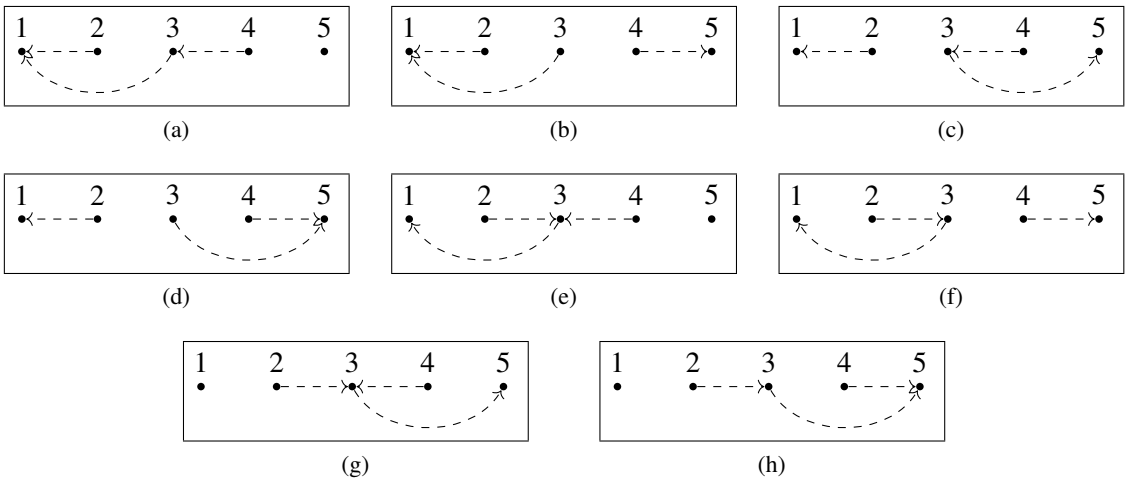


Fig. 6.6 Input graph: $C_5^{2\oplus}$; burning the edge $e(2,4)$. The *dashed* edges represent the *mapping* (absorb) edge. The *solid* edge represent the remaining edges after the iteration.

6.3.2.3 Generating function formulation

Let i be index of the selected edge, the expected number of component is then given by, for $L \geq 5$ ($N_0 = 0, N_1 = 1, N_2 = 1, N_3 = 1, N_4 = 1$):

$$N_L = \sum_{1 \leq i \leq L-2} N(i) = \begin{cases} 1/(L-2)N_{L-3} & i = 1, L-2 \\ 1/(L-2)(N_{i-1} + N_{L-i-2}) & 2 \leq i \leq L-3 \end{cases}$$

Multiply N_L by $L-2$ and similarly for N_{L-1} and subtracting $(L-3)N_{L-1}$ from $(L-2)N_L$ yields the following recurrence

$$(L-2)N_L = (L-3)N_{L-1} + 2N_{L-3}. \tag{6.9}$$

Let $\sum_n a_n x^n$ be the OGF that generates N_L. Multiply both sides by x^n and sum on $n \geq 5$ ($a_0 = 0, a_1 = 1, a_2 = 1, a_3 = 1, a_4 = 1$)

$$\sum_{n \geq 5} n a_n x^n - 2 \sum_{n \geq 5} a_n x^n = \sum_{n \geq 5} n a_{n-1} x^n - 3 \sum_{n \geq 5} a_{n-1} x^n + 2 \sum_{n \geq 5} a_{n-3} x^n$$

Transform the terms according to $G(x)$ gives

$$\begin{aligned} G'(x)(x - x^2) + x - x^3 - 2x^4 &= 2G(x)(1 - x + x^3) + x^2 - 3x^4 \\ G'(x)(x - x^2) - 2G(x)(1 - x + x^3) &= -x + x^2 + x^3 - x^4 \end{aligned}$$

The differential equation is then

$$G'(x) + G(x) \frac{2(x^3 - x + 1)}{x^2 - x} = x^2 - 1 \tag{6.10}$$

The integrating factor yields

$$e^{\int \frac{2(x^3-x+1)}{x^2-x}dx} = e^{x^2+2x+2ln(1-x)-2ln(x)} = (1-x)^2 e^{x^2+2x}/x^2$$

Thus $G(x)$ is given by

$$G(x) = \frac{\int \frac{(x-1)^2 e^{x^2+2x}}{x^2}(x^2-1)dx}{(1-x)^2 e^{x^2+2x}/x^2} = \frac{x^2}{(1-x)^2 e^{x^2+2x}} \int (x^4-2x^3+2x-1)\frac{e^{x^2+2x}}{x^2}dx$$

This equation is problematic because later on we need to evaluate $G(x)$ at $x=0$ to find the unknown constant of integration C. However, if we integrate by parts, then the function $\int e^{x^2+2x}/x^2 dx$ remains which is undefined at $x=0$. In which case, we can apply the method used in Section 3.3.3.2. That is, since the function in the integral involves the exponential function $e(x) = e^{x^2+2x}$, we differentiate $e(x)$ and manipulate $G(x)$ according to the known derivatives. More specifically, we have

$$\Big(\frac{e^{x^2+2x}}{x}\Big)' = (2x^2+2x-1)e^{x^2+2x}/x^2$$

Then we can manipulate $G(x)$ as follows

$$\begin{aligned}
G(x) &= \frac{x^2}{(1-x)^2 e^{x^2+2x}} \int (x^4-2x^3+2x-1)\frac{e^{x^2+2x}}{x^2}dx \\
&= \frac{x^2}{(1-x)^2 e^{x^2+2x}} \Big(\int (x^2-2x-2)e^{x^2+2x}dx + \int (2x^2+2x-1)e^{x^2+2x}/x^2 dx \Big) \\
&= \Big[\frac{x^2}{(1-x)^2 e^{x^2+2x}} \int (x^2-2x-2)e^{x^2+2x}dx \Big] + \frac{x}{(1-x)^2} \\
&= D(x)[A(x)+C] + \frac{x}{(1-x)^2}
\end{aligned}$$

where we have substituted $D(x) = \frac{x^2}{(1-x)^2 e(x)}$ and $A(x) = \int (x^2-2x-2)e(x)dx$. Next, we solve for C. It is seen that $G(0) = 0$ for any C. Thus we use other initial conditions of $G(x)$. We have the following derivatives up to second order, for each function of x in $G(x)$

$$D''(x) = \frac{d}{dx} \underbrace{\Big[\frac{-2x(x^3-x+1)}{e^{x^2+2x}(x-1)^3} \Big]}_{D'(x)} = \frac{2(2x^6-5x^4+6x^3+x^2-2x+1)}{e^{x^2+2x}(x-1)^4} \tag{6.11}$$

$$\Big[\frac{x}{(1-x)^2} \Big]'' = \frac{d}{dx} \Big[-\frac{x+1}{(x-1)^3} \Big]' = \frac{2(x+2)}{(x-1)^4} \tag{6.12}$$

Thus, for $G(x)$, we have the derivatives

$$G'(x) = D'(x)A(x) + A'(x)D(x) + C.D'(x) - \frac{x+1}{(x-1)^3} \tag{6.13}$$

$$G''(x) = D''(x)A(x) + D'(x)A'(x) + A''(x)D(x) + A'(x)D'(x) + C.D''(x) + \frac{2(x+2)}{(x-1)^4} \tag{6.14}$$

From (6.11) we see that $D'(0) = 0$ thus $C.D'(x) = 0$, further $G'(0) = 1 = a_1$ which satisfies the initial condition, however for any values of C. Next, we have $D''(0) = 2$ using (6.11). Hence for the second order derivative of $G(x)$ we have, using (6.14),

$$G''(0) = 2A(0) + 2C + 4 = 2 \quad \text{hence} \quad C = -1 - A(0).$$

Thus, the OGF is

$$G(x) = \frac{x^2}{(1-x)^2 e^{x^2+2x}}\Big(A(x) - A(0) - 1\Big) + \frac{x}{(1-x)^2} \tag{6.15}$$

6.3.2.4 Asymptotic of the coefficients of $[x^n]G(x)$

The first function of $G(x)$ has a simple pole of order 2 at $x = 1$, for which we have the following series expansion at $x = 1$

$$\begin{aligned} f(x) = \frac{x^2}{(1-x)^2} e^{-(x^2+2x)} &= \frac{1}{e^3}\Big(\frac{1}{(x-1)^2} + \frac{2}{x-1} + 1\Big)\Big(e^{-(x-1)^2} e^{-4(x-1)}\Big) \\ &= \frac{1}{e^3}\Big(\frac{1}{(x-1)^2} + \frac{2}{x-1} + 1\Big)\Big(1 - 4(x-1) + 7(x-1)^2 - \dots\Big) \\ &= \frac{1}{e^3}\Big(\frac{1}{(x-1)^2} - \frac{2}{x-1} + \dots\Big) \end{aligned}$$

Thus $[x^n]f(x) \sim \frac{1}{e^3}(n+3)$. It follows that

$$\begin{aligned} [x^n]G(x) &\sim \frac{n+3}{e^3} \times [A(1) - A(0) - 1] + (n+1) \\ &\approx (n+3)\Big(\big(\int_0^1 (x^2 - 2x - 2)e^{x^2+2x} dx - 1\big)/e^3 + 1\Big) \\ &\approx (n+3) \times ((-15.8283 - 1)/e^3 + 1) \approx (n+3) \times 0.16216 \end{aligned}$$

□

6.3.3 Combination of both processes: burning both main and odd/even edge

In this section, we consider a combination of *both* types of burn-edge process discussed in the previous sections. At each iteration we select an edge e at random. If e is a *main* edge it is burned by letting the two incident vertices *be absorbed* by any neighbours (but not each other, as stated in Section 6.3.1).

Else, e is burned by letting the two incident vertices be absorbed; the vertex inbetween is then also absorbed (see Section 6.3.2).

Denote by ℓ the total number of edges in the graph $C_L^{2\oplus}$ which has L vertices. Denote by ℓ_{main} and $\ell_{\text{e,o}}$ the number of *main* edges and *even/odd* edges, respectively. Hence, $\ell_{\text{main}} = L-1$ and $\ell_{\text{e,o}} = L-2$ and $\ell = 2L-3$. The algorithm then proceeds as follows. With probability ℓ_{main}/ℓ a *main edge* is *burned* generating $C_{i-1}^{2\oplus}$ and $C_{L-i-1}^{2\oplus}$. Else with probability $\ell_{\text{e,o}}/\ell$ an even/odd edge is *burned* generating $C_{i-1}^{2\oplus}$ and $C_{L-i-2}^{2\oplus}$.

Let the vertex set of $C_L^{2\oplus}$ be $V = \{v_1, v_2, \ldots, v_L\}$. Let M be the set of *main* edges and N the set of *even and odd* edges. Further, $M = \{m_1, m_2, \ldots, m_{L-1}\}$ where m_i is an edge which connects the pair of vertices (v_i, v_{i+1}). Similarly, $E = \{e_1, e_2, \ldots, e_{L-2}\}$ where each e_i connects the pair of vertices (v_i, v_{i+2}).

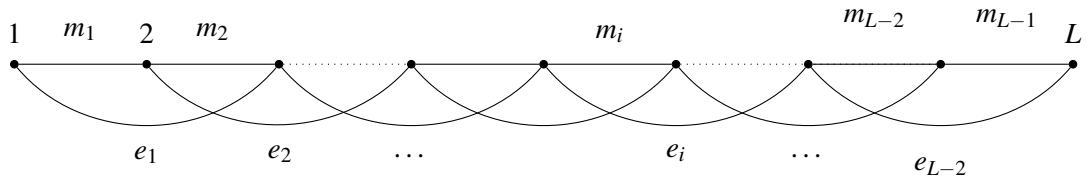


Fig. 6.7 The input graph $C_L^{2\oplus}$

Denote by N_L the expected number of components of the input graph. We have $N_0 = 0$; $N_1 = 1$; $N_2 = 2$ (since it contains one single main edge). The graph with three vertices has 2 main and 1 even/odd edge, it can be verified that $N_3 = 1/3(N_1) \times 3 = 1$; and $N_4 = 8/5$. The combined recurrence is given as follows. Let i, j be the index of the selected *main* or *even/odd* edge, for $L \geq 5$,

$$N_L = \sum_i M(i) + \sum_j E(j) \begin{cases} M(i) & \begin{cases} 1/(2L-3)N_{L-2} & i = 1, L-1 \\ 1/(2L-3)(N_{i-1} + N_{L-i-1}) & 2 \leq i \leq L-2 \end{cases} \\ E(j) & \begin{cases} 1/(2L-3)N_{L-3} & j = 1, L-2 \\ 1/(2L-3)(N_{j-1} + N_{L-j-2}) & 2 \leq j \leq L-3 \end{cases} \end{cases}$$

The above equation can be *condensed* nicely since the sum are symmetric for $M(i) : \sum_{2\leq i\leq L-2} N_{i-1} \equiv \sum_{2\leq i\leq L-2} N_{L-i-1}$; and for $N(j) : \sum_{2\leq j\leq L-3} N_{j-1} \equiv \sum_{2\leq j\leq L-3} N_{L-j-2}$. Multiply N_L by $(2L-3)$ and expand it we get

$$(2L-3)N_L = 2N_{L-2} + 2\sum_{i=1}^{L-3} N_i + 2N_{L-3} + 2\sum_{j=1}^{L-4} N_j = 2\sum_{i=1}^{L-2} N_i + 2\sum_{j=1}^{L-3} N_j$$

Carry out similarly for $C_{L-1}^{2\oplus}$, note this graph has two less edges compared to $C_L^{2\oplus}$, thus

$$(2L-5)N_{L-1} = 2\sum_{i=1}^{L-3} N_i + 2\sum_{j=1}^{L-4} N_j$$

Subtracting $(2L-5)N_{L-1}$ from $(2L-3)N_L$ yields

$$(2L-3)N_L = (2L-5)N_{L-1} + 2N_{L-2} + 2N_{L-3} \tag{6.16}$$

6.3.3.1 Generating function formulation

Let $G(x) = \sum_{n=0}^{\infty} a_n x^n$ be an odinary generating function that generates the sequence N_L, from (6.16) we get

$$(2n-3)a_n = (2n-5)a_{n-1} + 2a_{n-2} + 2a_{n-3} \quad (n \geq 5, a_0 = 0, a_1 = 1, a_2 = 2, a_3 = 1, a_4 = 8/5). \tag{6.17}$$

Multiply both side of (6.17) by x^n and sum on $n \geq 5$ yields

$$2\sum_n n a_n x^n - 3\sum_n a_n x^n = 2\sum_n n a_{n-1} x^n - 5\sum_n a_{n-1} x^n + 2\sum_n a_{n-2} x^n + 2\sum_n a_{n-3} x^n$$

Following the usual procedure, after some work, we get to the differential equation

$$G' + G\frac{2x^3 + 2x^2 - 3x + 3}{2x(x-1)} = \frac{x^2 + 2x - 1}{2} \tag{6.18}$$

Now the integrating factor

$$\mu(x) = \exp\Big(\int \frac{2x^3 + 2x^2 - 3x + 3}{2x(x-1)}\Big) = \exp\Big(x^2/2 + 2x + 2\ln(1-x) - 3/2\ln(x)\Big)$$

hence

$$\mu(x) = \frac{(1-x)^2 e^{x^2/2+2x}}{x^{3/2}}$$

Now for the denominator of $G(x)$

$$\int \frac{(1-x)^2 e^{x^2/2+2x}}{x^{3/2}} \frac{(x^2+2x-1)}{2} dx = \int \frac{(x^4 - 4x^2 + 4x - 1)e^{x^2/2+2x}}{2x^{3/2}} dx \tag{6.19}$$

The integral is quite problematic as it contains the exponential which is non-elementary. Furthermore, later on we will need to evaluate the function at $x = 0$ to find the constant of integration C. Thus, we want to *cancel* out the x in the denominator.

We apply the technique used in Section 3.3.3.2. Notice in (6.19) the denominator contains $x^{-3/2}$, thus its antiderivative should contain the factor $x^{-1/2}$. We proceed by, firstly, finding the derivative of the function $e(x) = e^{x^2/2+2x}/\sqrt{x}$. Then we rewrite $G(x)$ according to the known derivatives to

integrate by parts to, hopefully, *cancel* out the factor x in the denominator. More particularly, first we find the derivative

$$\frac{d}{dx}\frac{e^{x^2/2+2x}}{\sqrt{x}} = \frac{(2x^2+4x-1)e^{x^2/2+2x}}{2x^{3/2}}$$

Using the function above as a hint, we then rewrite the integral in (6.19) as follows

$$\begin{aligned}\int \frac{(x^4-4x^2+4x-1)e^{x^2/2+2x}}{2x^{3/2}}dx &= \int \frac{[x^4-6x^2+(2x^2+4x-1)]e^{x^2/2+2x}}{2x^{3/2}}dx \\ &= \underbrace{\int (\frac{1}{2}x^{5/2}-3\sqrt{x})e^{x^2/2+2x}dx}_{A(x)} + \int \frac{2x^2+4x-1}{2x^{3/2}}e^{x^2/2+2x}dx\end{aligned}$$

Let $e(x) = e^{x^2/2+2x}$, then $G(x)$ can be rewritten as

$$G(x) = \frac{A(x)+e(x)/\sqrt{x}+C}{\mu(x)} = \frac{x^{3/2}}{(1-x)^2e(x)}[A(x)+C] + \frac{x}{(1-x)^2} \tag{6.20}$$

where C is the unknown constant of integration. We now solve for C. For $A(x)$ we can expand the exponential function $e(x)$ and integrate by parts to obtain

$$\begin{aligned}A(x) &= \int (\frac{1}{2}x^{5/2}-3\sqrt{x})\Big(1+2x+\frac{5}{2}x^2+\frac{7}{3}x^3+\dots\Big)dx \\ &= \int \Big(-3\sqrt{x}-6x^{3/2}-7x^{5/2}-6x^{7/2}-\dots\Big)dx \\ &= -\Big(3\int \sqrt{x}dx+6\int x^{3/2}dx+7\int x^{5/2}dx+6\int x^{7/2}dx+\dots\Big) \\ &= -\Big(2x^{3/2}+\frac{12}{5}x^{5/2}+2x^{7/2}+\frac{4}{3}x^{9/2}+\dots\Big)\end{aligned}$$

We can also obtain a series expansion for the first term of $G(x)$ in (6.20) by convoluting the given three functions

$$\begin{aligned}\frac{1}{(1-x)^2}e^{-x^2/2}e^{-2x} &= \Big(1+2x+3x^2+4x^3+\dots\Big)\Big(1-\frac{1}{2}x^2+\frac{1}{8}x^4-\frac{1}{48}x^6+\dots\Big)\Big(1-2x+2x^2-\frac{4}{3}x^3+\frac{2}{3}x^4+\dots\Big) \\ &= \Big(1+2x+3x^2+4x^3+\dots\Big)\Big(1-2x+\frac{3}{2}x^2-\frac{1}{3}x^3-\frac{5}{24}x^4+\frac{3}{20}x^5+\dots\Big) \\ &= \Big(1+\frac{1}{2}x^2+\frac{2}{3}x^3+\frac{5}{8}x^4+\frac{11}{15}x^5+\dots\Big)\end{aligned}$$

Thus, we have, for the first function of $G(x)$

$$\begin{aligned}\frac{x^{3/2}}{(1-x)^2e(x)}A(x) &= -\Big(2x^3+\frac{12}{5}x^4+2x^5+\frac{4}{3}x^6+\dots\Big)\Big(1+\frac{1}{2}x^2+\frac{2}{3}x^3+\frac{5}{8}x^4+\frac{11}{15}x^5+\dots\Big) \\ &= -\Big(2x^3+\frac{12}{5}x^4+3x^5+\frac{58}{15}x^6+\dots\Big)\end{aligned}$$

It is seen from equation (6.20) the coefficients of $G(x)$ is given by the sum of three functions, specifically

$$[x^n]G(x) = [x^n]\frac{x^{3/2}A(x)}{(1-x)^2e(x)} + [x^n]C\frac{x^{3/2}}{(1-x)^2e(x)} + [x^n]\frac{x}{(1-x)^2} \tag{6.21}$$

Denote by

- b_n the n^{th} coefficients of $[x^n]\frac{x^{3/2}A(x)}{(1-x)^2e(x)}$; hence from equation (6.21) $b_0 = 0, b_1 = 0, b_2 = 0, b_3 = -2, b_4 = -12/5, b_5 = -3, \ldots$.
- c_n the n^{th} coefficients of $[x^n]C\frac{x^{3/2}}{(1-x)^2e(x)}$;
- d_n the n^{th} coefficients $[x^n]\frac{x}{(1-x)^2}$; we get $d_0 = 0, d_1 = 1, d_2 = 2, d_3 = 3, d_4 = 4, d_5 = 6, \ldots$.

Using the initial conditions of $G(x)$, specifically: $a_0 = 0, a_1 = 1, a_2 = 2, a_3 = 1, a_4 = 8/5$; comparing the coefficients of the above series we get

$$\begin{aligned}
[x^0]G(x) = a_0 &= c_0 = 0,\\
[x^1]G(x) = a_1 = c_1 + d_1 &= c_1 + 1 = 1,\\
[x^2]G(x) = a_2 = c_2 + d_2 &= c_2 + 2 = 2,\\
[x^3]G(x) = a_3 = b_3 + c_3 + d_3 = -2 + c_3 + 3 &= c_3 + 1 = 1,\\
[x^4]G(x) = a_4 = b_4 + c_4 + d_4 = -12/5 + c_4 + 4 &= c_4 + 8/5 = 8/5.
\end{aligned}$$

Hence, we can conclude that $c_n = 0$ for all n which implies that $C = 0$. In fact, the function $C\frac{x^{3/2}}{(1-x)^2e(x)}$ has fractional exponents while $G(x)$ does not have any c_n which supports this conclusion.

Finally, for the asymptotic of the coefficient of $G(x)$:

$$[x^n]G(x) \sim (n+1) \times (e^{-5/2}\int_0^1 (\frac{1}{2}x^{5/2} - 3\sqrt{x})e^{x^2/2+2x}dx + 1) \approx (n+1) \times 0.2538$$

□

6.4 Conclusion

In this chapter, we studied variations of the *burn-edge* algorithm in which *mappings* are done on a set of edges. We studied the basic algorithm on the cycle graph, then made some modifications and applied it to 2-circulant graphs. For each graph, we were able to estimate the expected number of components produced by the algorithms, using recurrence and generating functions.

To conclude the chapter, we carry out some experiments and present the empirical results. Particularly, we generate 2-circulant graphs with the number of vertices n varies from $1000, 2000, \ldots, 10000$. For each value of n, we execute each algorithm 10 times and take the averaged number of components. Figure 6.8 presents the *observed* number of components as a function of varying n. Further, using the

experimental number of components, we calculate

$$\text{Observed coefficient} = \frac{\text{Average number of components}}{n}$$

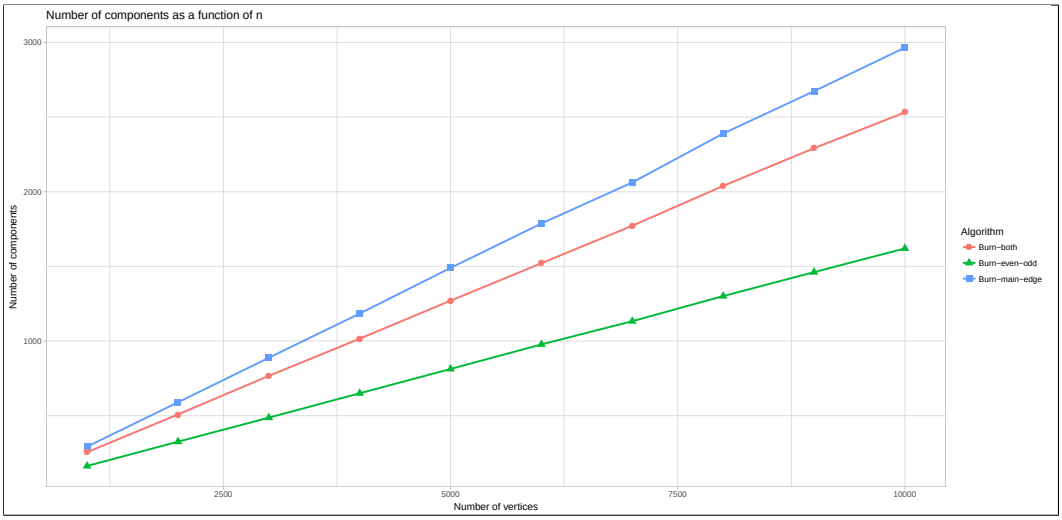
and compare it with the

$$\text{Expected coefficient} = \lim_{n\to\infty} \frac{E[\text{number of components}]}{n}$$

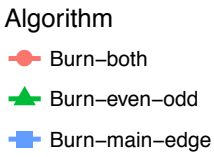
Table 6.1 shows the above coefficients.

Burn-edge Variation	Expected	Observed
Burn main edges	0.2970	0.2965
Burn even/odd edges	0.16216	0.162
Burn both types	0.2538	0.2531

Table 6.1 Table of Summary: the expected and observed values of the number of components produced by variations of burn-edge algorithms on 2-circulant graph.

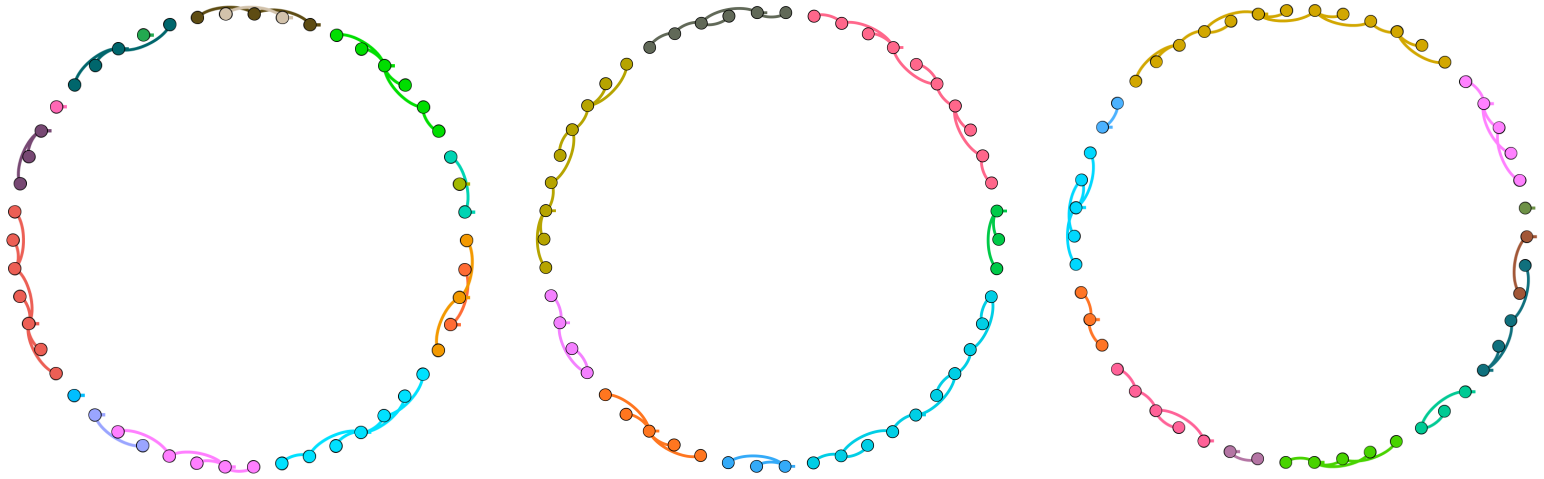


(a)



(b) Legends

Fig. 6.8 The figure shows the *experimental number of components* of each algorithm on 2-circulant graph as a function of the *number of vertices*. In this experiment, the number of vertices n is increased from $1000, 2000, \ldots, 10000$. Each line corresponds to a burn-edge variation.



(a) Burn main edges (b) Burn even/odd edges (c) Burn both types

Fig. 6.9 A visualisation of the burn-edge algorithm on a 2-circulant graph of 50 vertices. The vertices are *colour coded* according to the resulting components. The roots can be identified as the vertices with little knots on top (this is due to the limitation of the software). The numbers of components for each algorithm are, from left to right, 16, 8 and 12. Visually, the graph in the middle seems to have the most consistent colours distribution. On the contrary, the figure on the left seems to contain several small components *mixed* together.

Another observation is that in Figure (a) it is quite common to find two roots next to each others. In Figure (b), the roots are often found *inside* theirs components. A possible explanation is given by the base cases of each algorithm. When *burning only main edges*, it is possible to form two roots in an iteration, see Figure 6.3. Whereas, when *burning even/odd edges*, only one root is formed at a time, see Figure 6.4. In Figure (c), it seems we have a mixture of both behaviours.

Part III

Triangle fragmentation with application to clustering

Chapter 7

Literature review of graph clustering

In Part I and Part II, we introduced and studied graph fragmentation algorithms. In general, the algorithms take an input graph and *fragment* it, or in other words they partition the input into a number of smaller components. Recently, there is an ever increasing interest in clustering problem in general and graph clustering or community detection in particular [20]. The objective of this task is to divide a set of objects in such a way that objects in the same group i.e. *a cluster* are more similar (in some sense) to each other as opposed to those in other clusters [7] [49]. The desired output of such problem is then a set of clusters, each consists of objects which are highly similar. In some sense, the output of the clustering problem are fragments of the input. Thus, we are interested in investigating the applicability of fragmentation algorithms on such problem. This is the main objective for the remaining chapters of this thesis. We begin by briefly reviewing the literature on clustering problem.

Clustering is not a specific type of algorithm, instead, it is a general problem to be solved [55]. The term *'cluster'* is commonly used as a noun to refer to the set of objects or sometimes as a verb i.e. to partition [18]. On the other hand, *'clustering'* is often used to refer to the output of the algorithms i.e. the set of clusters containing the objects of interest. Commonly, there are two types of clusterings: *hard clustering* in which each object belongs to at most one cluster and *soft clutering* in which each object belongs to a cluster with a degree of certainty, say a probability.

A clustering can be achieved by various algorithms that differ significantly in their understanding of what defines a cluster and how to efficiently find them [18]. Popular notions of clusters include: small distances between members, dense areas of the data points space, statistical distributions or vertices connectivity [55]. Notably, there lacks a general consensus on the definition of a *cluster*, which is one of the reasons why there are so many clustering algorithms [18]. The notion of cluster varies significantly, depending on different models, definitions and settings employed by researchers. Some popular models of cluster include: *centroids-model* (k-means algorithm [47]); *density-based model* (OPTICS algorithm [26]); *probability distribution-based model* (Expectation-Maximisation algorithm [14]), *graph-based model* (see following sections) and so many more.

As there is no agreed definition of cluster, there is also no objectively *correct* clustering algorithm [18]. Therefore, the most appropriate clustering algorithm for a particular problem are often evaluated

and chosen *experimentally*. Hence, the general procedure for testing algorithms is to applying it to a clustering problem whose solution is known a-priori [20]. In which, the problem and the solution are often regarded as the *benchmark* and the *ground-truth*, respectively. The solutions delivered by the algorithm are then compared with the ground-truth for theirs similarities, using some specific measures/indices. Finally, the obtained measurements are compared against the results of the current state-of-the-art algorithms. Thus, the essential tool-set for studying clustering problem consists of: *benchmarks*, *measures* and *algorithms in comparison*.

In this study, we are interested in two specific settings of clustering: *graph* and *geometric*. In a *graph* setting, the objects of interest are vertices and edges. The objective here is to divide the vertices into groups or *communities* so that there is a higher degree of *connectivity* for vertices *within* a communities, and lower for vertices *between* different communities [43]. In *geometric* settings, the objects of interest are data points embedded in space. The similarity of the objects can be measured using theirs *distance* i.e. points in close proximity are considered similar. Thus, the objective in this setting is to divide points into groups such that the distance between cluster members are minimised.

Chapter's structure. In this chapter, we focus our attention on the literature on *graph clustering problem*. This is to prepare for the following Chapters 8, 9 and 10 which investigate the application of fragmentation algorithms on graph clustering. Chapter 11 and 12 apply fragmentation on geometric data in which the former serves as a preparation chapter on the geometric clustering problem.

The remaining sections of this chapter are structured as follows. Section 7.1 defines the *graph clustering/community detection* problem. Section 7.2 introduces some popular benchmark graphs which consists of real-world networks and computer-generated networks. Section 7.3 introduces measurements for evaluating the performance of clustering algorithms. Section 7.4 briefly introduces some of the popular graph clustering and community detection algorithms.

The terms: graph clustering and community detection; networks and graphs; groups, partitions, communities and clusters are used interchangeably.

7.1 Definition and motivation

Graph clustering is the task of dividing vertices of the graph into *groups/clusters/communities* i.e. sets of vertices, so that there is a higher degree of *connectivity* for vertices *within* a cluster, and lower for vertices *between* different clusters.

The problem arises naturally by the observation that many real life networks e.g. social, biochemical and information networks have the tendency for vertices to divide into groups with dense connection *within* and sparser *between* them [20]. For instances, people tend to form groups in their social or work environments. In fact, a similar behaviour is found to exist in dolphins (see Figure 7.2 page 121, [38]). In protein-protein interaction networks, proteins belong to the same functional group interact very frequently with each other [20], and therefore the interest in identifying communities arises naturally when one studies the structures of these networks. Such communities often correspond to units which share the same features or functions, and can have local properties that are very different

from their properties at the level of the entire network. We take a closer look at this phenomenon by examining some real-world networks in the following section.

7.2 Networks

7.2.1 Real world networks

These graphs were obtained by researchers in various studies. The vertices are usually the subjects of the studies, and the edges are interactions between them. In these cases, the *perceived ground-truth communities* are observed by the authors. The formation of these communities is caused by various factors i.e. human psychological behaviour, geographical factors, or based on the author's conclusions. The point is, although, these partitions can be considered the *ground-truth* communities, it is the result of some decision making process, either naturally or depending on human psychology. Therefore, the ground-truth communities might not always yield the best results or the correct answers.

7.2.1.1 The karate club.

Constructed by W. Zachary [64] (see Fig. 7.2, page 121), the network represents a social relationship between members of a karate club at an American university. It consists of 34 vertices and the edges connect members who were observed to interact with each other outside the club's activities. Eventually, due to some dispute between the teacher and the administrator, the club split into two separate smaller clubs centred around these two individuals. The separation is then considered as the ground-truth communities in this graph.

7.2.1.2 The football clubs.

Introduced in [23], the network is a representation of Division I League of the United States' college American-football teams. The vertices represent the teams and edges represent regular-season games between the two teams. The teams are divided into 12 conferences, each contains $8-12$ teams. Games occur more frequently between *intra-conference* teams. Inter-conference games are less frequent and depends on geographic factors.

It should be noted that there is an *independent conference*. Of which, their games are more geographically dependant i.e. it is more convenient to organise games with geographically closer team. Therefore, its intra-conference games are actually much less frequent compare to other conferences; thus having an abnormally inter-intra edge ratio (see Figure 7.2 page 121, grey vertices).

7.2.1.3 Political books.

This graph was compiled by Krebs during the Presidential Selection in the United States in 2004. The network consists of 105 vertices representing books on American politics, bought from Amazon.com. The edges represent *co-purchasing* of books by the same buyers. The ground-truth community in this

graph is added later by Newman in [23]. The partition of the network is according to each book's stated or apparent political alignment i.e *liberal* or *conservative*. With an exception for a small number of books that were explicitly bipartisan or centrist, or had no clear affiliation, which are considered as *neutral*.

7.2.1.4 Social network of Bottlenose Dolphins.

The graph was constructed by Lusseau et al, by observations of a community of 62 bottlenose-dolphins over a period of seven years from 1994 to 2001. The vertices in the network represent the dolphins and the edges represent associations between each dolphin pair which occur more often than expected by chance [38]. Later, the dolphins separated into two groups after a dolphin left for a period of time. Interestingly, the separation is quite transparent from the interactions (edges) between its members; which can be seen in Figure 7.2, page 121.

7.2.2 Computer generated networks

7.2.2.1 Planted ℓ partition model.

This class of graphs is a popular benchmark for testing graph clustering. It is otherwise known as the *planted ℓ partition model* originally introduced in [9] and later considered by the authors from [23]. Thereafter became a standard benchmark graph for testing graph clustering algorithm. It is generated as follows.

First, we generate a *global* graph G. Next, we divide the vertices of G into ℓ subsets of equal size. We then add an extra layer of edges to connect vertices within each subset. Thus, the subgraph induced by each of these subsets is denser than the global graph G. We call these hidden subgraphs the *planted partitions*. The purpose of the model is to test whether an algorithm can recover correctly these partitions hidden within the global graph G.

In more detail, the global graph G is often generated by, first, construct a random graph i.e. $G^{(1)}(n,p)$. Next, we divide the vertex set V into ℓ subsets of equal sizes i.e. $S = \{S_1, \ldots, S_\ell\}$ where each S_i, *a hidden partition*, is a disjoint set of vertices i.e. $S_i \cap S_j = \emptyset$ and $S_1 \cup \cdots \cup S_\ell = V$. Let $|S_i| = m = n/\ell$, we add an additional edge layer to the partitions by considering each as a random graph $G^{(2)}(m,q)$. If $q = 0$, then the graph G is just a normal random graph, and hence there is no partition to recover. For $q > 0$, the hidden partitions become denser and G has more apparent *cluster structure*.

7.2.2.2 Lancichinetti-Fortunato-Radicchi benchmark graph

It is argued by Lancichinetti et al in [34] that various properties of random graphs are far different from real networks e.g. degree distribution. Therefore, the application of random graphs in the *planted ℓ partition model* does not provide a good reflection of real life network that possess community

structure. In order to improve this, the authors proposed a model that yields a *power law* distribution for the degree and cluster size.

The graph, denoted as LFR graph, requires the following main parameters:

$$\text{Params} = \{N, [k_{\min}, k_{\max}], k, \mu, \tau_1, \tau_2, [c_{\min}, c_{\max}]\}$$

Where

- N: number of vertices;
- $[k_{\min}, k_{\max}]$: minimum and maximum degree of any vertex;
- k: the average degree of vertices in the graph;
- $\mu \in [0, 1]$: the *mixing parameter*;
- τ_1, τ_2: the power law parameters, the former for generating degree sequence and the latter for community sizes;
- $[c_{\min}, c_{\max}]$: community size range.

An important parameter is μ, which decides the number of edges of a vertex to be *internal* and *external* edges. For instance, consider a vertex v with degree $d(v)$, then $(1-\mu)d(v)$ would be the number of *internal edge* i.e. connecting to vertices in the same community.

The graph generation process is provided below, as described in the original paper [34]:

1. Each vertex is given a degree taken from a power law distribution with exponent τ_1. The extremes of the distribution k_{min} and k_{max} are chosen such that the average degree is k [34]. Note that the author does not provide a detailed explanation of this sampling process.

2. Similar to the degree sequence, each community is given a size sampled from a power law distribution with parameter τ_2. So that each community size falls within $[c_{\min}, c_{\max}]$ and all sizes sum up to N. Denote by $s_{\min}$ and $s_{\max}$ the *sampled minimal* and *maximal community sizes*; these two are selected so that:

   $$s_{\min} > k_{\min} \qquad\qquad s_{\max} > k_{\max}.$$

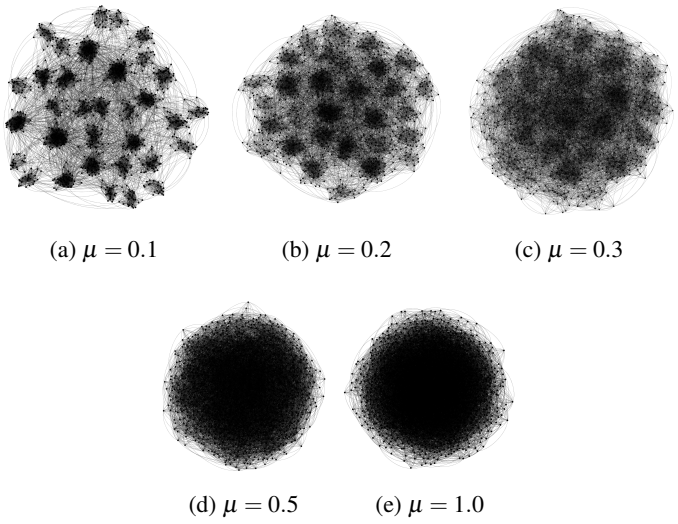
   This is to ensure that a vertex of any degree can be included in at least one community i.e. it has enough edges to connect to vertices in the same/different community in all cases, because the edge rewiring is done after this step. For example, if $\mu = 0$ then each vertex is only connected to vertices in the same community. Thus, each vertex must be put in a community such that $s_{\min} > k_{\min}$ and $s_{\max} > k_{\max}$ so that there are sufficient number of vertices.

3. For each vertex v with degree $d(v)$, it is determined that a fraction of its edges i.e. $(1-\mu)d(v)$ are *internal edges*; the remaining are *external edges*.

4. The community assignment proceeds as follows. At the beginning, all vertices are *homeless*. In the first iteration, a vertex v is assigned to a random community. If the community size is greater than the vertex internal degree i.e. $(1-\mu)d(v)$, it is accepted. Otherwise, it is assigned to another community. In the case that a community is *full*, the *queried vertex* replaces a *previously accepted* vertex chosen at random. The latter is kicked out and becomes homeless. The procedure carries on there is no more homeless *vertices*.

5. The final work is to rewire the edges. The *configuration model* is used to preserve the sampled degree sequence. The principal is that for each vertex v, its edges are *broken* into *stubs* or half-edges. Next, a fraction $(1-\mu)d(v)$ of the stubs are connected to other stubs of vertices in *its community*; and the rest to stubs in *other communities*. Note that it is possible to have *loops* i.e. vertex connects to itself or double-edge, in which case the graph is rejected and re-generated.

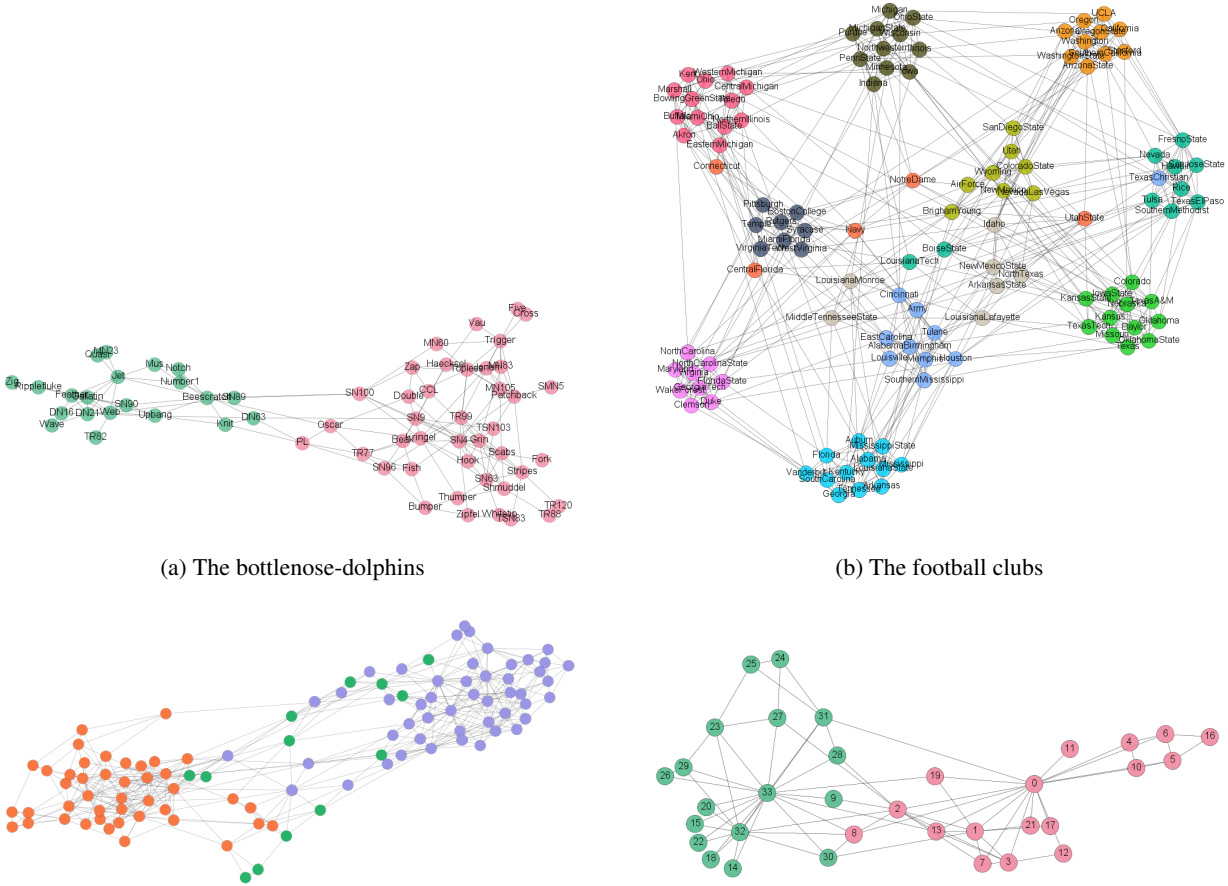
It is seen that the *mixing parameter* μ determines the *community structure* of the graph. For instances, $\mu = 0$ then the communities are simply components and if $\mu = 1$ then there is no community.



(a) $\mu = 0.1$ (b) $\mu = 0.2$ (c) $\mu = 0.3$

(d) $\mu = 0.5$ (e) $\mu = 1.0$

Fig. 7.1 A visualisation of the LFR-benchmark graph with $n = 1,000$. The cluster sizes vary in range $[20-50]$ and the value of the *mixing-parameter* μ increases from 0.1 to 1.0. It is seen that the cluster-structure of the graph gradually disappears as μ increases.



(a) The bottlenose-dolphins

(b) The football clubs

(c) The political books network

(d) The karate club

Fig. 7.2 The input social networks. For each vertex in each network, its colour corresponds to its ground-truth community. (a) The network bottlenose-dolphins, the two communities are quite easily identifiable given there are very few edges joining the two. (b) The football teams, the colour represents each team's conference. There are teams appear to be more closely connected to teams in another conference. This is due to the games are geographically dependant. (c) Books in *orange* align with *liberal*; *purple* with *conservative* and *green* are *neutral*. (d) Vertex 0 is *the instructor*, vertex 33 is *the administrator*. Visually, the centrality of these two vertices are apparent. One can also roughly identify the separation of such network, with some difficulties at vertices $13, 2, 8$ which lie between the two groups.

7.2.2.3 Remarks

Both the planted-ℓ-partition and LFR benchmark graphs are widely used as tools for evaluating clustering algorithms [20]. While the former is more useful for obtaining analytical results, the latter is preferred for empirical evaluation. Arguably, a disadvantage of the LFR is that it is less understood analytically compares to the GN/planted-partition which employs the extensively-studied random graph model.

Generating LFR graphs requires a wide variety of parameters e.g. mixing parameter μ, degree range, community size range, power law parameters, etc. (see Table 7.1). However, there seems to be no *agreed, default set* of the parameters. Thus it is uncertain how different the networks are given different set of parameters; or whether the structure of a graph undergoes any changes by altering any given parameter, etc.

We briefly examine, experimentally, some properties of the LFR graph. Particularly, we are interested in the clustering coefficient (this metric will be introduced in Section 7.3.2.2 page 126) of the graph. This is because the clustering coefficient is related to the number of triangles in graphs, which is a fundamental property of community.

To generate LFR graphs, we use the software package provided by the authors, available at [2]. The parameters used are given in Table 7.1. Note that we only consider non-overlapping communities. Also, the average clustering coefficient parameter is omitted because it is not always possible to achieved the desired input as noted in [34].

Parameter	Symbol	Value
Number of vertices	N	10,000
Average degree	k	15
Mixing parameter	μ	[0.1,0.7]
Power law exponent parameter for degree sequence	τ_1	2
Power law exponent parameter for size distribution	τ_2	1
Communities sizes	[minc,maxc]	[10,50];[50,100];[100,150];[150,200] [200,250];[250,300];[300;350]
Number of overlapping vertices	on	0
Average clustering coefficient	C	omitted

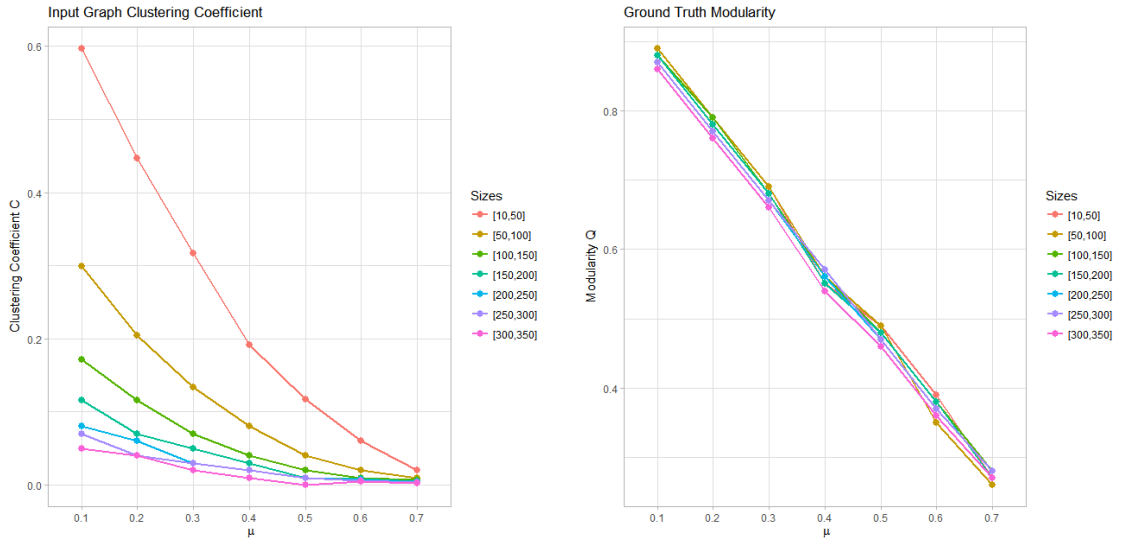
Table 7.1 Parameters for generating LFR graphs

Figure 7.3 (a) shows the experimentally obtained clustering coefficient $\overline{C}$ (see section 7.3.2 below for details) as a function of the mixing parameter μ. Each line corresponds to a different parameter of *community size* e.g. red - $[10,50]$, brown - $[50,100]$, etc.

Generally, it is seen that $\overline{C}$ decreases as μ increases. Furthermore, we also noticed that $\overline{C}$ varies according to cluster size. Specifically, the larger the cluster sizes, the lower the clustering coefficient of the graph. For instance, for $\mu = 0.1$, the clustering coefficients are: $0.6, 0.3, 0.18$ for the cluster

sizes: $[10,50]$ (red), $[50,100]$ (brown) and $[100,150]$ (light green) respectively. This might be due to that there is not enough edges (as we set $k = 15$) given the number of vertices in a community to form triangles.

Beside the clustering coefficient, Figure 7.3 (b) presents the modularity of the ground truth, which appears to decrease linearly with μ and be independent from cluster sizes. This makes intuitive sense, since μ is the only factor that determines the number of *internal/external* edges. For instance, with $\mu = 0.1$, for every vertex 90% of its edges are connected to vertices within the same community. Hence, for any cluster the proportion of internal/external edges are 9/1, resulting in very high modularity Q.



(a) LFR: clustering coefficient

(b) LFR: modularity of ground-truth communities

Fig. 7.3 Clustering coefficients of LFR graphs; and modularity of the ground truth communities. The plots are for a graph with $n = 10,000$ vertices. Each line corresponds to a different *community size* parameter e.g. $[10,50]$ red, etc.

7.3 Evaluation metrics

In this section, we introduce the criteria for evaluating the performance, particularly the accuracy, of the algorithms. Recall that each input graph has a *ground truth* community structure [44] which can be considered the *answer* to each problem. Hence, we need to derive a criterion to establish how *similar* the output is to these apriori communities.

Consider the ground-truth communities. As each vertex belongs to *exactly* one community, let the truth communities be the set $\mathscr{T} = \{T_1, T_2 \dots T_i\}$, where each T_i is a *disjoint* set of vertices e.g. $T_1 \cap T_2 = \emptyset$; and hence $T_1 \cup \dots \cup T_i = V$. Next, let the set of output communities be $\mathscr{C} = \{C_1, C_2 \dots C_j\}$, where j may or may not equal i; and each set $C_j = \{v, w, \dots\}$ is a *disjoint* set of vertices $C_1 \cup \dots \cup C_j =$

V. Then, we would like a function $f(\mathscr{T},\mathscr{C}) \to x \in R$ that evaluates the similarity x between the two sets.

7.3.1 Pair counting measures

A popular set of measures: **Rand** [49], **Jaccard** [28] and **Adjusted Rand Index** (ARI) [27] are based on the principal of *pair counting*. It is computed by counting the number of pairs of vertices which are classified in the same and different clusters in the two given sets.

More specifically, given the two sets: $\mathscr{T}$ - the ground-truth and $\mathscr{C}$ - the results, we can consider this as *before* and *after* sets. We count [49]:

- a, the number of pairs of elements that are in the same set in $\mathscr{T}$ and in the same set in $\mathscr{C}$;
- b, the number of pairs of elements that are in different sets in $\mathscr{T}$ and in different sets in $\mathscr{C}$;
- c, the number of pairs of elements that are in the same set in $\mathscr{T}$ and in different sets in $\mathscr{C}$;
- d, the number of pairs of elements that are in different sets in $\mathscr{T}$ and in the same set in $\mathscr{C}$;

		$\mathscr{C}$: Same	$\mathscr{C}$: Diff
$\mathscr{T}$	Same	a	c
	Diff	d	b

Table 7.2 An illustration of all possible pair placements.

An interpretation of the above procedure is as follows. Suppose the total number of vertices is n, then there are $\binom{n}{2}$ possible pair of vertices. The number a then counts the number of *true positive* i.e. pairs that are partitioned in the same clusters before and after. The number b counts the number of *true negative* i.e. pairs that are in different clusters before and after. The number of errors is counted by c and d. More specifically, c is the number of *false negative* i.e. the pairs which are in same ground-truth clusters but partitioned into different clusters; and d is the number of *false positive* i.e. vice versa of c.

The *Rand* index is given by the fraction of *correct placements* over all possible pair placements

$$\text{Rand} = \frac{a+b}{a+b+c+d} = \frac{a+b}{\binom{n}{2}}. \tag{7.1}$$

However, a weakness of the Rand index is that a and b are equally weighted. In cases where there are a large number of differences, b becomes enormous and overshadows a,c,d. And hence *Rand* tends to 1, thus giving a false sense of accuracy. In such case, an alternative can be the *Jaccard* Index:

$$\text{Jaccard} = \frac{a}{a+c+d}. \tag{7.2}$$

Since Jaccard index takes into consideration only the *correct* pairs over the summation of *correct* pairs and the number of disagreements between the partitions, it gives a more accurate similarity measure between the two partitions.

Another alternative is the *Adjusted Rand Index* (ARI), which is the corrected-for-chance version of the Rand index. The ARI assumes the generalised *hypergeometric distribution* as the model of randomness, i.e., the $\mathscr{T}$ and $\mathscr{C}$ partitions are picked at random such that the number of objects in the classes and clusters are fixed [27]. The ARI is then given by:

$$\text{ARI} = \frac{Index - ExpectedIndex}{MaxIndex - ExpectedIndex},$$

where the indices are computed using the formulas introduced by Hubert and Arabie in [27], see Section C.1.2 for details.

Since Rand, Jaccard and ARI are principally similar; we often present results using **ARI** since it is commonly regarded as the best among the three. ARI scores range from $[-1,1]$ with the higher score indicates the closer match, with 1 being an *exact* match.

7.3.2 Cluster quality

Previously, we have introduced the pair-counting measures, which are used for comparing the resulting clusters to the apriori communities. However, as mentioned the apriori communities are not necessarily the *correct or best* way to partition the input graph. Therefore, it would be useful to have an alternative criterion to evaluate the *goodness* of the resulting communities.

There exists various quality functions for such purpose. Arguably, two of the most popular are: *modularity* and *clustering coefficient*. Each measure bases its definition of *goodness* on a different concept of *connectivity*: the former on edge density, and the latter on the number of closed triangles. Below we discuss the two.

7.3.2.1 Modularity

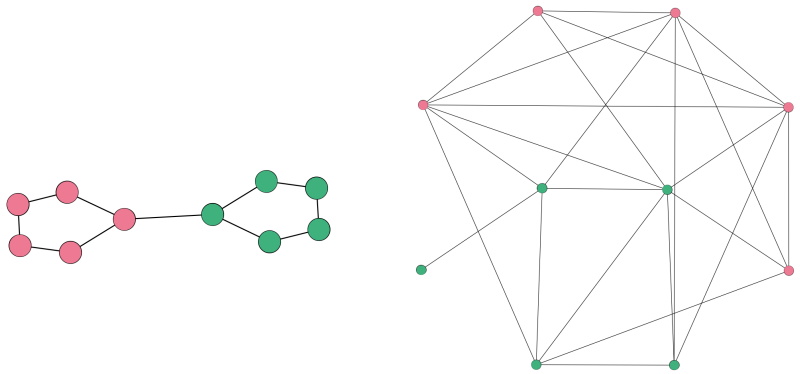
Originally introduced by Girvan and Newman in [45], the modularity Q measures the strength of the partitions of a graph into modules. It hypothesises that a network with high modularity has high-edge density between vertices within each cluster (high *intra-edge*) but sparser between different clusters (low *inter-edge*). It is based on the idea that a random graph is not expected to have any cluster structures. So the existence of clusters is revealed by the comparison between the *actual edge density* with the *expected edge density* chosen from a random graph model. In this sense, the concept of modularity is quite similar to ARI's.

In brief, the measure is calculated as follows (for more details, see Section C.1.1). Let C be the set of output clusters. Suppose we select a community $c \in C$, let e_c be the number of edges with both ends in c and d_c is the degree sum of vertices in c. Let m be the total number of edges in the graph.

Then

$$Q = \sum_{c \in C} \left(\frac{e_c}{m} - \left(\frac{d_c}{2m} \right)^2 \right). \tag{7.3}$$

The range of value for Q lies in $[-0.5, 1)$, with a higher value than 0 suggests the number of edges within groups exceeds the number expected on the basis of chance. In practice, it is noted that a good value for community graph lies in the range from 0.3 to 0.7 [45].



(a) Q = 0.4; C = 0 (b) Q = 0.08; C = 0.5

Fig. 7.4 Small example graphs, the communities are indicated by its colours. The left graph has modularity $Q = 0.4$ and $C = 0$. The right graph is a *small world network* which has a high clustering coefficient $C = 0.5$ however the best partition produces only $Q = 0.08$. The reason for the difference between Q and C is that the graph on the left has no triangle, thus $C = 0$. The graph on the right has many triangles but they are evenly distributed, thus no clustering and $Q = 0.08$.

7.3.2.2 Clustering Coefficient

Introduced by Watts and Strogatz in [58] to detect whether a graph is a *small world* network. The measure is based on the idea that a pair of vertices which have a common neighbour is likely to be connected. There are two versions of this metric: *global clustering coefficient* and *local clustering coefficient*. Here we only discuss the latter.

The local clustering coefficient of a vertex determines how close its neighbours are to being a clique. Let there be a graph $G(V,E)$ and suppose we select a vertex $v \in V$. Let $d(v)$ be its degree and $N(v)$ be its set of adjacent neighbours, thus $d(v) = |N(v)|$. The local clustering coefficient of v $C(v)$ is defined by the number of edges between the adjacent neighbours of v, over the number of all possible edges between them. Since we are dealing with simple graph, the number of possible edges is $\binom{d(v)}{2} = d(v)\{d(v)-1\}/2$. Thus

$$C(v) = 2 \times \frac{|e(u,w) : u,w \in N(v); e(u,w) \in E|}{d(v)\{d(v)-1\}}, \tag{7.4}$$

where $e(u,w)$ denote the edge between the pair u, w. Note that $e(u,w) = e(w,u)$ which counts as one single edge. And hence $C(v) \in [0,1]$ with 0 indicates there is no mutual neighbour between v and any its adjacent vertex; and 1 indicates the $N(v)$ is a clique i.e. every neighbour connects to every other vertex in the neighbourhood. It also follows that $C(v)$ is undefined for vertex v with $d(v) = 0$ or 1.

The average clustering coefficient is then computed by taking the mean of the local clustering coefficients i.e.

$$C(G) = \frac{1}{|V|} \sum_{v \in V} C(v). \tag{7.5}$$

7.3.2.3 Remarks

The two metrics discussed here are based on two different concepts of *connectivity*. Placed in the context of clustering, Q is more *dynamic* while C is *static*. In the sense that the modularity Q gives different values for different partitions of the vertices of the graph into communities. Whereas C remains constant for each input graph, and therefore, Q would arguably be a more suitable metric for comparing different outputs of clustering algorithms. In fact, *modularity* seems to be the preferred criteria for most of the clustering algorithms which are introduced below.

The two concepts are related in the sense that a large number of triangles in the cluster increases the number of edges inside it i.e e_c in equation (7.3). Thus a graph can have both high Q and C. For example, consider the left example from Figure 7.4. If we add more edges to make the two communities, internally, *complete* graphs, then we would get $Q = 0.45$ and $C = 0.89$. Furthermore, Table 7.3 shows that the real world networks introduced earlier have relatively high values of both Q and C.

	Q	C
Karate	0.358	0.571
Dolphins	0.378	0.303
Polbook	0.414	0.244
Football	0.553	0.403

Table 7.3 Q & C of real networks introduced in Section 10.1.

7.4 Algorithms in comparison

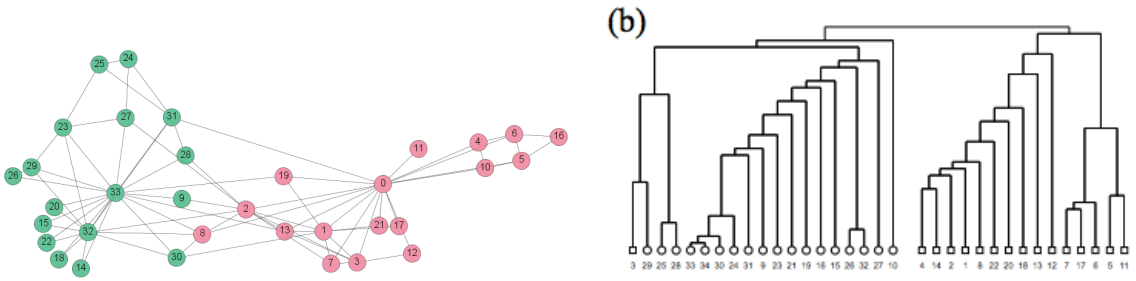
In this section, we give brief descriptions of the community-detection algorithms. For its detailed description, see Section C.2.

7.4.1 GN: Edge-Betweenness Centrality Clustering

Given a vertex v, its *vertex-betweenness* is a measure of the influence of a vertex over the flow of information in a graph. It is defined as the *number of shortest paths that pass through v* [21]. Girvan and Newman extend the definition of betweenness to *edges* in [23]. That is for a given edge e its *edge-betweenness* is defined as the number of shortest paths between each pair of vertices that pass along it. For any pair of vertices, if there is more than one shortest path, each path is assigned an equal weight such that the total weight adds up to unity.

The authors observe that if a network contains communities that are only sparsely connected by a few inter-group edges, then all shortest paths between different communities must use one of these few inter-group edges. Therefore, the edges connecting communities will have *high edge betweenness*. Consequently, by removing these edges, the groups would be separated thus revealing the underlying community structure of the network.

The algorithm proceeds by greedily calculating the edge betweenness for every edge in the graph, then removing the one with highest score, until no edge remains. The output of the algorithm is a *dendogram* (see e.g. Figure 7.5), which is built by the order of the removed-edges. The height of each *branch* is given by the edge-betweenness value of the corresponding edge. To produce the result partitions, the tree is cut at a suitable height to obtain the desired number of communities.



(a) The karate club (b) Result dendogram

Fig. 7.5 The dendogram of the karate network, produced by the edge-betweenness centrality clustering algorithm. The figure is taken from the original paper [23].

7.4.2 CNM: Fast Greedy Modularity Optimisation.

The algorithm was introduced by Clauset et al in [8]. It employs a *greedy modularity optimisation* strategy. Starting with n communities, with each community contains a single vertex, in each step two communities whose merger produces the largest increase in Q are joined. The process is repeated for $n-1$ steps. At the end, a *dendogram* is produced, which represents the community structure of the network.

7.4.3 Louvain Method for Large Networks

This algorithm is based on a *greedy modularity optimisation* approach. The algorithm, named *Louvain method*, was introduced by Blondel et al [5] and was shown to achieve one of the best running times. It consists of *two phases*, which are repeated iteratively: first, for each vertex, find the neighbour which yields the largest gain in modularity. Next, the two are then merged as one *super vertex*. The algorithm then continues until there is no further possible merging.

The efficient factor of the algorithm is due to the fact that ΔQ can be easily calculated when putting (or removing) a vertex v in a community C using equation (7.3). Finally, it is pointed out in [20] that the result is dependant on the *order of selection* of vertices in step one.

7.4.4 Infomap

This algorithm was introduced by Rosvall and Bergstromin in [50]. The authors tackle graph clustering by employing an approach from *information theory*. The idea is to describe a graph more efficiently by using less information than the traditional way of using the full adjacency matrix. The objective is to optimally compress the information required to describe the information on the network, by exploiting regularities in the flow of information using *random walks* as follows [50]:

1. Each vertex in the graph is given a unique name using an efficient method of encoding i.e. Huffman encoding;

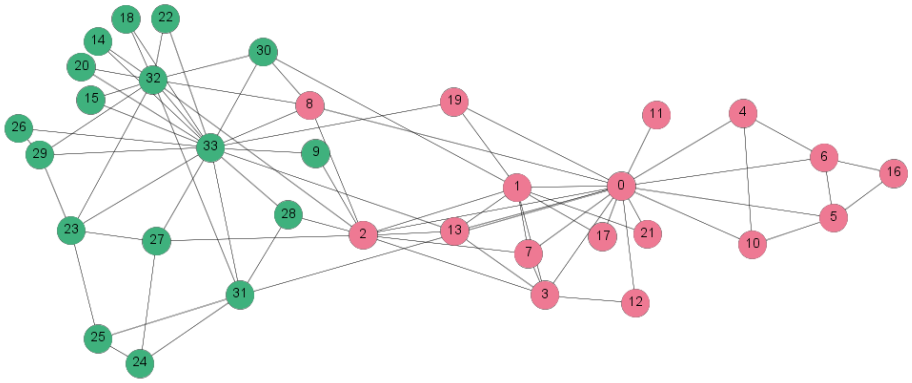
2. Let the *walk* takes some number k steps. For each step taken, its *traversed-vertex* is recored using its unique coding. The whole walk can then be represented by a sequence of vertices' names obtained from the previous step.

3. Having the walk's names, the problem of partitioning is then posed as a coding problem i.e. to find a partition that minimises the expected description length of the coding. The minimisation is achieved through incorporating greedy search with simulated annealing.

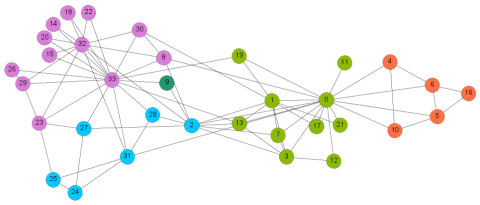
At the end of the algorithm, the result is an efficient coding of the trajectory of the random walk. The coding consists of two-level description in which the higher-level are the unique names of the major clusters, which is used to decompose the input graph into respective clusters.

7.5 Conclusion

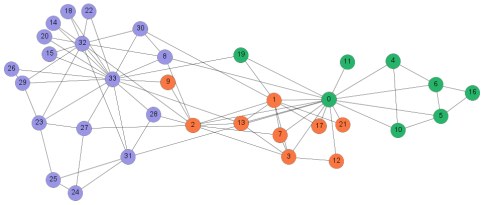
In this chapter, we introduced the essential tool set for studying *graph clustering* problem. The tool set consists of: a set of *benchmark graphs* for testing; various measurements for evaluating the performance of the algorithms and finally some popular algorithms for drawing comparison. This toolset will be used to study and evaluate the performance of graph fragmentation algorithms as a clustering tool in the following chapters. The literature covering the field of clustering in general and graph clustering/community detection in particular is immense. As such, this review could only cover some basic and necessary tools which would be useful for this study. For an in-depth literature review, we recommend [20].



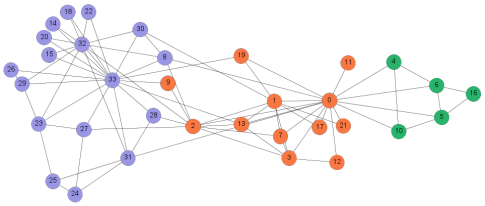
(a) Ground-truth communities of the Karate network



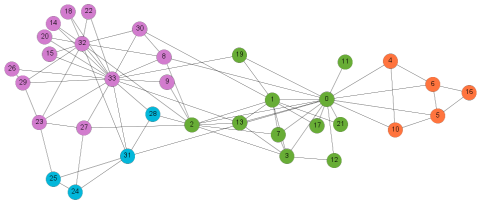
(b) GN



(c) CNM



(d) Infomap



(e) Louvain

Fig. 7.6 The above figures show the communities of the *Karate network*. Figure (a) shows the *ground-truth community structure* of the network. Figure (b) - (e) show the communities produced by the reviewed algorithms, which are labelled accordingly.

Chapter 8

Introduction to the k-MXT algorithm

Imagine the following scenario: in a large group, a task is given that is to form arbitrary groups, and the set method is for each individual to join with k others. Imagine that for an individual, the group consists of people whom they acquainted with and those who are not. And the question is how an individual would decide? The hypothesis here is that an individual is more likely to select those that he/she is *most familiar with*. Another hypothesis is given to quantify the *most familiar factor* is the number of *mutual friends* between two individuals.

To put it in graph terminology, given a simple, undirected graph $G(V,E)$, we want to partition the graph into smaller subgraphs such that each subgraph is desirably *having tighter connections among its vertices*. There can be various definitions of high connectivity, but inspired by the *thought scenario* above, we lean toward the definition of having *closed-triangles*.

The closest metric/measure that is defined in terms of *triangles* is *clustering coefficient* which quantifies how close its neighbours are to being a clique. The local clustering coefficient of a vertex in a graph is defined as follows. Let $N(v)$ be the set of adjacent vertices of a vertex v, that is

$$N(v) = \{u : e(v,u) \in E(G)\}$$

then $|N(v)| = d(v)$ is the degree $d(v)$ of vertex v. The local clustering coefficient $C(v)$ for vertex v is then given by the proportion of *actual edges* between the vertices within its adjacent neighbours divided by the number of *edges that could possibly exist* between them. Since there are $\binom{d(v)}{2} = \frac{d(v)(d(v)-1)}{2}$ possible edges, we need to count the number of existing edges between every pair of v's neighbours. For every unique edge $e(u,w)$ that exists such that $u,w \in N(v)$, we have a *closed triangles* $\{v,u,w\}$; then let $\triangle(v)$ be the number of such triangles, we get

$$C(v) = \frac{2\triangle(v)}{d(v)(d(v)-1)}$$

we will use these definitions as a basis for the following algorithm, which is called k-MaX-Triangles (k-MXT).

8.1 The k-MXT algorithm

Number of triangles from a vertex perspective. The algorithm proceeds as follows. For each vertex $v \in V$, we examine $N(v)$. For each neighbour $u \in N(v)$ we count the number *closed triangles* formed on the edge $e(v,u)$ i.e. the triplet $\{v,u,w\}$ such that all three edges (v,u), (u,w) and (v,w) are present, that is:

$$\triangle(v,u) = |w \in N(v) : \{v,u,w\} \text{ is a closed triangle}|$$

Alternatively, $\triangle(v,u)$ is cardinality of the set intersection of $N(v)$ and $N(u)$

$$\triangle(v,u) = |N(v) \cap N(u)|$$

Number of triangles from an edge perspective. The above description of the number of triangles is taken on the perspective of each vertex in G. Alternatively, a description that focuses on each edge can be given as follows. Let G be an *edge-weighted* graph. Every edge $(v,u) \in E(G)$ is given a *weight* which is the number of common neighbours of the two incident vertices, that is

$$w(v,u) = \triangle(v,u) = |N_i \cap N_j| = |\{w : w \in N(v) \wedge w \in N(u)\}|$$

Selecting k highest triangles-weighted neighbours. For each vertex v, we select the k neighbours of v which have the highest weight. We call this set the k-Max-Triangles neighbours of v. In case of tie, candidates are selected at random. If for any vertex its degree is at most k i.e. $d(v) \leq k$, then all adjacent vertices are selected.

Creating the subgraph H. Finally, for each v and a selected vertex u i.e. u is in the k-Max-Triangles set, we add the edge (v,u) to an empty graph H. Note that the edge (v,u) is naturally directed, but we can choose to ignore its orientation.

After every vertex made its selection, at most $|V| \times k$ edges are added to H. Furthermore, if we ignore the edges orientation, double edges and self-loop, then H is an undirected subgraph of G, induced by these *edges* given by the selections of the vertices, $V(H) = |V|$ and $|E(H)| \leq |V| \times k$. With k small, the subgraph H is much sparser compared to G and hence typically contains a larger number of components. We call the components in H the *fragments* of G. The fragments are the objects of interest. For instance, if the objective is to partition the input graph G then the set of fragments is the output. Furthermore, the *membership* of each vertex v_i is determined by the component it belongs to in H, which can be examined by any *connected components* algorithm.

8.2 Pseudocode

The following pseudocode illustrates the k-MXT algorithm.

Algorithm 6: k-MXT

Input : graph G

Initialisation: empty graph H, $V(H) = V(G)$ **foreach** *edge* $(v,u) \in V$ **do**
  $w(v,u) = \triangle(v,u)$

end

foreach *vertex* $v \in V$ **do**

  Select k-MXT(v) **foreach** *vertex* $u \in k\text{-}MXT(v)$ **do**
    $E(H) = E(H) \cup e(v,u)$

  end

end

Output: H

8.3 Example

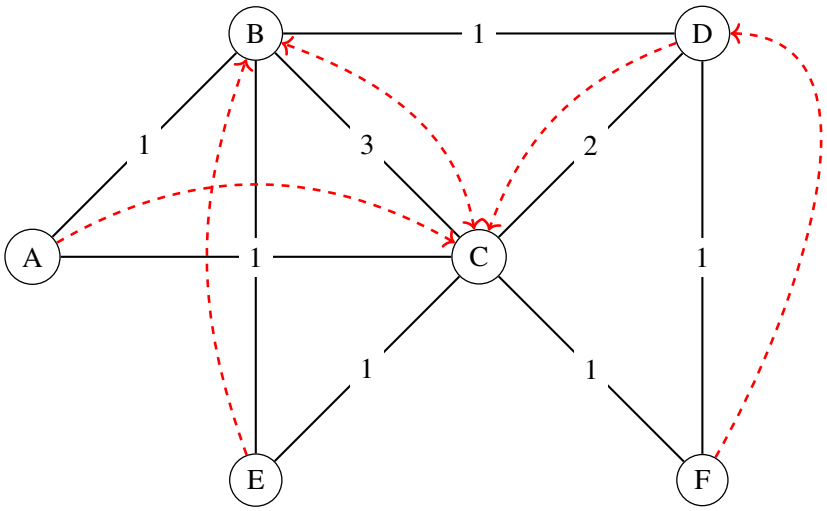


Fig. 8.1 An example graph G with its weighted edges i.e. $w(B,C) = 3$ since $N_B \cap N_C = \{A,E,D\}$, thus $w(B,C) = |N_B \cap N_C| = 3$. The *red* edges show the subgraph H, which is the 1-MXT graph.

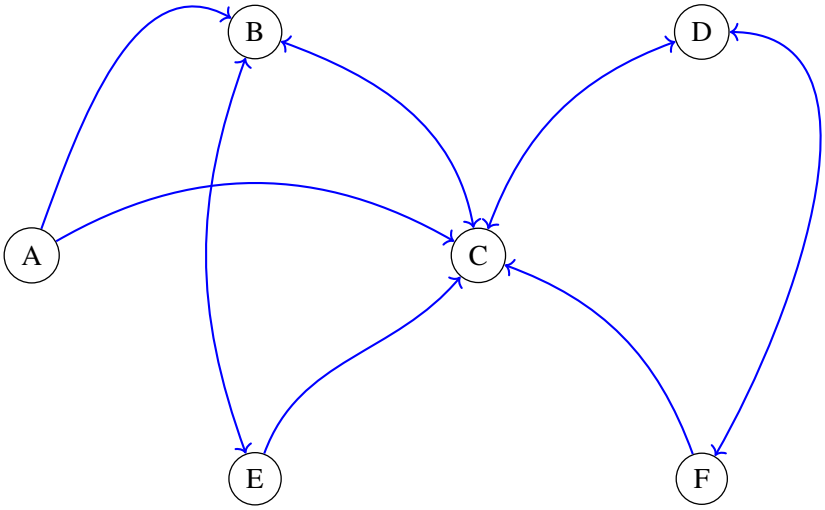


Fig. 8.2 *2-MXT graph*. Compare to 1-MXT, the new edges $(A,B),(E,C),(F,C)$ are visibly drawn, while $(B,C),(C,D),(D,F)$ are double-edges.

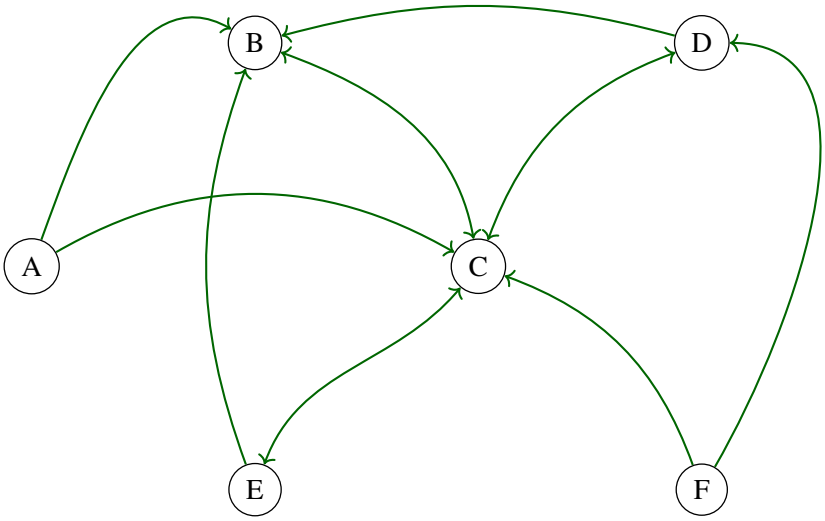


Fig. 8.3 *3-MXT graph*. Since there are only two vertices C and D which have out-degree 3 in H, the only visible different edge compare to 2-MXT is (D,B), since edge (C,E) is now a double-edge.

Chapter 9

The hidden block model

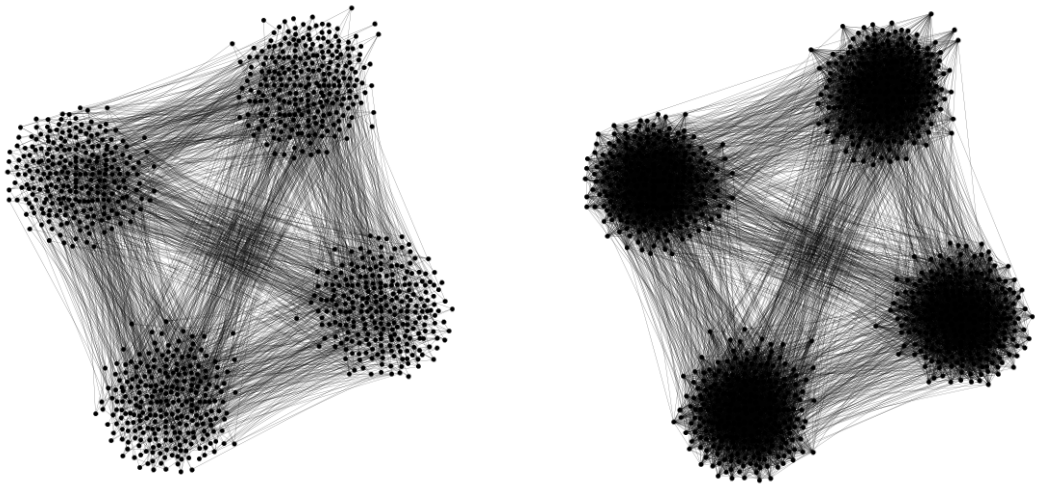
To test the functionality of the k-MXT algorithm, we use the *hidden-block model with planted partition* which was introduced in Section 7.2.2.1 (page 118). We review the model's synopsis below.

First, we generate a *global* graph G. We then divide the vertices of G into ℓ subsets of equal size - the hidden partitions. An extra layer of edges is then added to connect vertices within each partition. Thus, the partition induced by each of these subsets is denser than the global graph G. The purpose of the model is to test whether an algorithm can recover correctly these partitions hidden within the global graph G.

In more detail, the global graph G is often generated by, first, construct a random graph i.e. $G^{(1)}(n,p)$. Next, we divide the vertex set V into ℓ subsets of equal sizes i.e. $S = \{S_1, \ldots, S_\ell\}$ where each S_i, *a hidden partition*, is a disjoint set of vertices i.e. $S_i \cap S_j = \emptyset$ and $S_1 \cup \cdots \cup S_\ell = V$. Let $|S_i| = m = n/\ell$, we add an additional edge layer to the partitions by considering each as a random graph $G^{(2)}(m,q)$. If $q = 0$, then the graph G is just a normal random graph, and hence there is no partition to recover. For $q > 0$, the hidden partitions become denser and G has more apparent *cluster structure*.

As introduced in previous section, the output of k-MXT algorithm is a set of subgraphs of the input graph, which we call *fragments*. These fragments are created by having each vertex *point* to its k neighbours with which it shares the most common neighbours. If each partition is sufficiently dense then it should contain many triangles, and hence all vertices would be *pointing* to neighbours within its partition. This is the motivation for our analysis in this section.

By its nature k-MXT can only be used on graphs with an adequate number of *triangles*. Thus we begin by considering the probability p in relation to the number of triangles in the graph G.



(a) $q = 0$ (b) $q > 0$

Fig. 9.1 Hidden block graph example with $n = 4,000$, $\ell = 4$ and $m = 1,000$.

9.1 The application of k-MXT on the planted ℓ partitions model

9.1.1 The probability p such that there is an adequate number of triangles in the graph

The expected number of triangle in a random graph $G(n, p)$ is

$$E[\#\text{ triangles}] = \binom{n}{3} p^3$$

Let $p = n^{-c}$ this is $\Theta(n^{3-3c})$. If $c \leq 2/3$ then the expected number of triangles becomes at least linear with the graph size and hence significant globally.

Lemma 8. *Let $G = G(n, p)$. Let X be the expected number of triangles in G associated with vertex v. Let $p = \frac{1}{n^c}$ where $c < 2/3$. Then*

$$E[X] = \binom{n-1}{2} p^3 \approx \frac{n^{2-3c}}{2} \to \infty$$

and

$$Pr(X = 0) = O(\frac{1}{n^{2-3c}} + \frac{1}{n^{1-c}}) \to 0$$

Take a vertex v, let T_v be the number of triangles associated with v. A triangle i.e. the triplet $\{v, u, w\}$ associated with v exists with probability p^3, and there are $\binom{n-1}{2}$ possibilities to select the remaining pair of vertices. Although the existence of triplets are not exactly independent e.g. $\{v, u, w\}$

and $\{v,u,x\}$ [46], by linearity of expectation

$$E[T_v] = \binom{n-1}{2} p^3 = \frac{(n-1)(n-2)}{2} p^3 \approx \frac{n^{2-3c}}{2} \tag{9.1}$$

by substituting $p = n^{-c}$. Thus, $c = 2/3$ is a threshold, in the sense that if we choose $c > 2/3$, equivalently $p < 1/n^{2/3}$, then $E[T_v]$ tends to 0 as n tends to infinity. And on the contrary, with $p > 1/n^{2/3}$, the expected number of triangles tends to infinity. In which case, however, it does not imply that every vertex has an associated triangle. Nevertheless, using the second moment method, we can show that almost surely there is a triangle associated with each vertex.

Let X_i be an indicator random variable for each triangle associated with vertex v, and let $X = \sum X_i$ be its sum, then

$$Var(X) = E[X^2] - (E[X])^2 \leq E[X^2] \tag{9.2}$$

If we substitute $t = E[X]$ in the Chebyshev inequality (see Theorem 13, section B.1) then

$$Pr(|X - E[X]| \geq E[X]) \leq \mathrm{Var}(X)/(E[X])^2$$

which implies that $Pr(X = 0) \to 0$ given that $\mathrm{Var}(X)/(E[X])^2$ tends to 0. Following which, we will now look at the variance of X.

By definition, the variance of a sum of random variables is

$$\mathrm{Var}(X) = \sum_i \mathrm{Var}(X_i) + 2\sum_{i,j} \mathrm{Cov}(X_i, X_j) \tag{9.3}$$

We have, from (9.2),

$$\mathrm{Var}(X_i) \leq E[X_i^2] = p^3$$

For the covariance of two variables X_i and X_j, notice that if the two triangles do not share a common edge, then X_i and X_j are independent and $\mathrm{Cov}(X_i, X_j) = 0$. On the other hand, if the two triangles share a common edge, then $E[X_i X_j] = p^5$. Thus, by definition

$$\mathrm{Cov}(X_i, X_j) \leq E[X_i X_j] = p^5$$

For the number of pair of triangle sharing a common edge that associate with v, there are $n-1$ ways to form the edge and $\binom{n-2}{2}$ to choose the remaining vertices of the triangles, thus there are $(n-1)\binom{n-2}{2} = 3\binom{n-1}{3}$ possibilities. Substituting these into (9.3), we get

$$\mathrm{Var}(X) \leq \binom{n-1}{2} p^3 + 6\binom{n-1}{3} p^5 \leq n^2 p^3 + n^3 p^5$$

And therefore

$$\frac{\mathrm{Var(X)}}{(E[X])^2} \leq \frac{n^2 p^3 + n^3 p^5}{n^4 p^6} = \frac{1}{n^{2-3c}} + \frac{1}{n^{1-c}}$$

Thus $Pr(X = 0) \to 0$ as $n \to \infty$ as desired. □

9.1.2 Expected number of triangles on a given edge

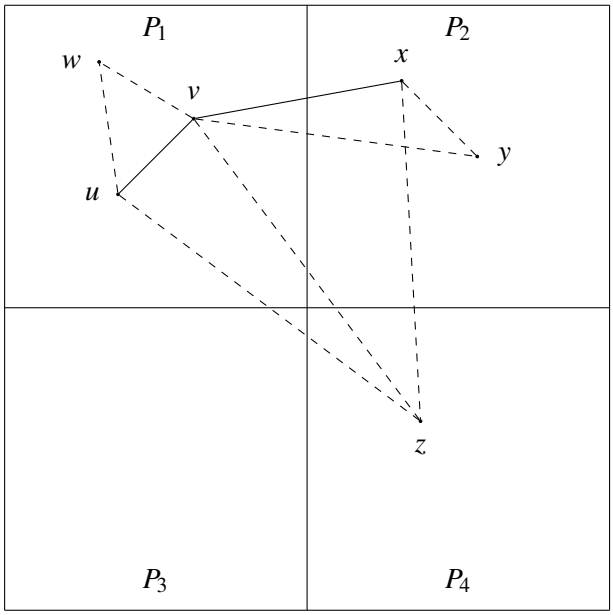


Fig. 9.2 Illustration of triangles in a planted 4 partitions.

As illustrated above, the graph is generated by first generating a *global* graph G as a $G(n,p)$ with the addition of some denser subgraphs $P_1,\ldots,P_\ell$, where each P_i is a $G(m,q)$ with $m = n/\ell$. In this section, we are interested in the expected number of triangles which are formed by *intra edges* i.e. within partition and *inter edge* i.e. across partition.

Take a vertex $v \in P_1$ and select a vertex u also in P_1, the probability that the edge (v,u) is not present in the *global* graph G is $1-p$. Further, the probability that the same edge is not present in the *hidden partition* P_1 is $1-q$. Hence, let r be the probability that such edge exists, then

$$Pr(\text{an intra edge } (v,u)) = r = 1-(1-p)(1-q) = p+q-pq \approx p+q \tag{9.4}$$

if $pq = o(1)$.

For a pair of vertices v,u in the same partition, let $X_{vu} = X(intra, vu)$ be the number of vertices w with edges (w,u) and (w,v). For a pair of vertices v,x in different partitions, let $Y_{vx} = Y(inter, vx)$ be the number of vertices w with edges (w,v) and (w,x). We note that these variables do not depend on the presence or absence of the edge (v,u) or (v,x).

Lemma 9. *Let $\ell = 4$ and let $q = 2\beta p$. Let (v,u) be an intra-cluster edge and let (v,x) be an inter-cluster edge. Then*

$$E[X(intra,vu)] = np^2(1+\beta+\beta^2)$$
$$E[Y(inter,vx)] = np^2(1+\beta)$$

Consequently, let $E[X(intra)] = \alpha E[Y(inter)]$ then

$$\alpha \approx 1 + \frac{\beta^2}{1+\beta}$$

Assuming the edge (v,u) is present, for each vertex w of the $\binom{m-2}{1}$ remaining vertices in P_1 the probability that the triplet (v,u,w) is formed is r^2. Similarly, for each vertex $z \notin P_1$ the probability the triplet (v,u,z) is present is probability p^2. Thus, the expected number of triangles on the intra edge (v,u), or equivalently, the expected weight of an intra edge is

$$\begin{aligned} E[X(intra,vu)] &= \binom{m-2}{1} r^2 + 3mp^2 \approx m(r^2+3p^2) \\ &= mp^2\big((1+2\beta)^2+3\big) = np^2(1+\beta+\beta^2) \end{aligned} \tag{9.5}$$

by substituting $r \approx p+q = p(1+2\beta)$, and $m = n/\ell = n/4$ thus $n = 4m$.

For an edge (v,x) in which $x \notin P_1$ i.e. $x \in P_2$, the probability that a triplet (v,x,y) with y being in either P_1 or P_2 is rp. And for any other vertex z that is not in the same partition as x and y, the triplet (v,x,z) exists with probability p^2. Thus, the expected number of triangles on an inter edge (v,x) is

$$E[Y(inter,vx)] = 2\binom{m-1}{1} rp + 2\binom{m}{1} p^2 \approx 2mp(r+p) = np^2(1+\beta) \tag{9.6}$$

Putting $E[X(intra)] = \alpha E[Y(inter)]$, it follows that

$$\alpha = 1 + \frac{\beta^2}{1+\beta} \tag{9.7}$$

□

On the other hand, if we plug in the full expression for r from (9.4) i.e. $r = p+q-pq = p(1+2\beta(1-p))$, then

$$\alpha = \frac{r^2+3p^2}{2p(r+p)} = \frac{\big(1+2\beta(1-p)\big)^2+3}{4\big(1+\beta(1-p)\big)} = \frac{1+\beta(1-p)+\beta^2(1-p)^2}{1+\beta(1-p)}$$

which simplifies to

$$\alpha = 1 + \frac{\beta^2(1-p)^2}{1+\beta(1-p)} \approx 1 + \frac{\beta^2}{1+\beta} \tag{9.8}$$

as $p \to 0$, this agrees with (9.7).

9.1.3 Threshold for β to avoid selecting inter partition edges.

Let $G(V,E)$ be a graph generated by the hidden-block model with a global graph $G^{(1)}(n,p)$ and hidden partitions $G^{(2)}(m,q)$. Assuming p is fixed, $p = 1/n^c$ with $c < 2/3$ to ensure sufficient a number of triangles, and $q = 2\beta p$. As β increases, more *internal edges* are added in each hidden partition. Consequently, when each partition becomes sufficiently dense, given any vertex the number of *internal triangles* associated with that vertex should far exceed the number of *external triangles*. Thus k-MXT should produce an accurate partition.

In this section, we are interested in the value for β, which with high probability there would be no cross-partition edge included. We assume $\ell = 4$, and hence the number of vertices in each partition is $m = n/\ell = n/4$.

For convenience, we denote the weight of an intra edge (v,u) and inter edge (v,x) as X and Y instead of $X(intra, vu)$ and $Y(inter, vx)$ as seen previously. From equation (9.6), we have

$$E[Y] = np^2(1+\beta) = n^{1-2c}(1+\beta)$$
$$E[X] = np^2(1+\beta+\beta^2) = n^{1-2c}(1+\beta+\beta^2)$$

For $p = n^{-c}$ and $c > 1/2$, $E[Y] = O(n^{-c}) \to 0$ as $n \to \infty$. As this case is of little interest, we exclude it and assume $p > n^{-1/2}$.

Let us examine Y. Chernoff's concentration inequality (see Theorem 14, section B.2) states that

$$Pr\{|Y - E[Y]| \geq \delta E[Y]\} \leq e^{-\delta^2 E[Y]/3} \quad (0 < \delta < 1) \tag{9.9}$$

We choose $\delta = \sqrt{\omega/E[Y]}$ with $\omega = 9\log n$. Note that, Chernoff requires $0 < \delta < 1$, which will be examined shortly below. Then, substitute δ into the Chernoff inequality we get

$$Pr\{|Y - E[Y]| \geq \delta E[Y]\} \leq e^{-\delta^2 E[Y]/3} = e^{-\omega^2/3} = e^{-3\log n} = n^{-3} \tag{9.10}$$

As there are at most $\binom{n}{2}$ edges (v,x) to consider, the probability any edge (v,x) deviates is $O(n^2 \times n^{-3}) = O(1/n)$.

Similarly, we choose $\widehat{\delta} = \sqrt{\omega/E[X]}$. Then X is bounded as

$$Pr\{|X - E[X]| \geq \widehat{\delta} E[X]\} \leq e^{-\widehat{\delta}^2 E[X]/3} = e^{-3\log n} = n^{-3} \tag{9.11}$$

Hence the probability that any edge (v,u) (with v and u in the same partition) deviates is also $O(1/n)$.

It is seen that Y is unlikely to be more than $(1+\delta)E[Y]$; and X is unlikely to be less than $(1-\widehat{\delta})E[X]$. Therefore set

$$\delta E[Y] + \widehat{\delta} E[X] \leq E[X] - E[Y]$$

which simplifies to

$$\sqrt{\omega} = 3\sqrt{\log n} \leq \frac{E[X]-E[Y]}{\sqrt{E[X]}+\sqrt{E[Y]}} = \sqrt{E[X]}-\sqrt{E[Y]} = n^{1/2-c}\left(\sqrt{\beta^2+\beta+1}-\sqrt{\beta+1}\right) \tag{9.12}$$

9.1.3.1 For $c < 1/2$.

Recall that Chernoff's inequality requires that $\delta = \sqrt{\omega/E[Y]} < 1$, or equivalently,

$$3\sqrt{\log n} < \sqrt{E[Y]} = n^{1/2-c}\sqrt{1+\beta}$$

which implies $c < \frac{1}{2} - O\left(\frac{\log\log n}{\log n}\right)$ or that $p = n^{-c} \geq 3\sqrt{\log n/n}$. Thus (9.12) yields

$$3n^{c-1/2}\sqrt{\log n} < \sqrt{\beta^2+\beta+1}-\sqrt{\beta+1} \tag{9.13}$$

as $c < 1/2$ the left hand side is $O(n^{-\varepsilon}) \to 0$ for $n \to \infty$. The right hand side is approximately at least linear for $\beta > 0$. Thus the inequality (9.13) is true for any $\beta > 0$.

Thus we have the following

Theorem 8. *For $p \geq 3\sqrt{\log n/n}$ and $q = 2\beta p$ for any $\beta > 0$ constant and $k \geq 1$. The probability that k-MXT wrongly chooses an inter-partition edge tends to zero as $O(1/n)$.*

9.1.3.2 Remark

In Section 9.3 below we set up experiments with $n = 10^4$ and $p = 1/\sqrt{n} = 0.01$. It appears that separation also occurs with $p = 1/\sqrt{n}$ hence $c = 1/2$; for $q/p = 2\beta = [6,8]$ hence $\beta = [3,4]$, see Figure 9.6 (page 148). Thus, we examine this case below.

9.1.3.3 For $c = 1/2$.

With $c = 1/2$, thus $p = 1/\sqrt{n}$. Let $\beta = \omega$ with $\omega \to \infty$, the number of intra and inter triangles are then

$$E[X] = \beta^2+\beta+1 \approx \omega^2; \qquad E[Y] = \beta+1 \approx \omega.$$

Using the Chernoff inequality for large deviations we have, for Y,

$$Pr(Y \geq \alpha E[Y]) \leq \left(\frac{e}{\alpha}\right)^{\alpha E[Y]} = \left(\frac{e}{\alpha}\right)^{\alpha\omega} \tag{9.14}$$

Whereas for X we can reuse (9.11)

$$Pr(X \leq (1-\delta)E[X]) \leq e^{-\delta^2 E[X]/3} = e^{-\delta^2\omega^2/3} \quad (0 < \delta < 1) \tag{9.15}$$

thus provided that $\omega = 3\sqrt{\log n}/\delta$ for any $0 < \delta < 1$, the above probability tends to 0 or $O(1/n^3)$.

In order to avoid selecting inter-partition edge, it must be

$$\alpha E[Y] < (1-\delta)E[X] \quad \text{thus} \quad \alpha < (1-\delta)\omega$$

Let $\alpha = c\omega$ where $c < 1-\delta$. Then consider the assertion for the probability for the upper bound in (9.14)

$$(\frac{e}{\alpha})^{\alpha\omega} < \frac{1}{n^3} \quad \text{equivalently} \quad \alpha\omega(\log\alpha - 1) > 3\log n$$

Substitute in α we get

$$\alpha\omega(\log\alpha - 1) = c\omega^2[\log(c\omega) - 1] = c\frac{9\log n}{\delta^2}\left[\log(c\frac{3\sqrt{\log n}}{\delta}) - 1\right] > 3\log n$$

Provided $c > \delta^2/3$ this is true for $n \to \infty$ and δ, c constant. It follows that (9.14) is $O(1/n^3)$ as required. Further, we require that

$$\delta^2/3 < c < 1-\delta \quad \text{or} \quad \delta^2 + 3\delta - 3 < 0,$$

solve for $\delta > 0$ we get

$$\delta < \frac{1}{2}(\sqrt{21} - 3) \tag{9.16}$$

□

In our experiments (see Section 9.3 for more details), we set $n = 10^4$ and $p = 1/\sqrt{n} = 0.01$ and observed that separation occured. The experimental results (see Figure 9.5) show that the separation occurs with $q/p = 2\beta = [6,8]$ hence $\beta = [3,4]$.

Whereas using the above calculation with $p = 1/\sqrt{n}$ then separation occurs, using (9.16), when q/p is at least

$$\frac{q}{p} \approx \omega = \frac{3}{\delta}\sqrt{\log n} = \frac{3}{0/79}\sqrt{\log 10^4} = 11.5.$$

9.1.4 Intra-inter edge weight ratio

The $G(n,p)$ model [16] is known to have a rather low clustering coefficient i.e. number of closed triangles. Nevertheless, it would still be of interest to keep track of the number of triangles formed by the respective intra or inter edges. For such, we calculate the ratio of the *edge-weight* as denoted previously to have an indication of whether there are more triangles within or across the hidden partitions.

For a pair of vertices v,u in the same partition, denote by $X_{vu} = X(intra, vu)$ be the number of vertices w with edges (w,u) and (w,v). For a pair of vertices v,x in different partitions, let $Y_{vx} = Y(inter, vx)$ be the number of vertices w with edges (w,v) and (w,x). Then the triangle ratio of vertex v is

$$Ratio(v) = \frac{\sum_u X(intra, vu)}{\sum_x Y(inter, vx)}$$

thus the ratio is < 1 indicates that there are more closed triangles formed by inter-edges from v; and > 1 indicates otherwise.

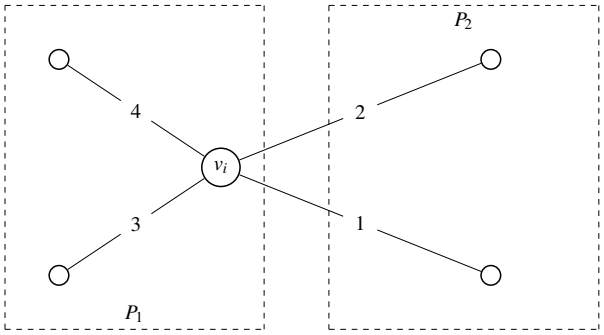


Fig. 9.3 Suppose v_i has the above weighted-edges, then $Ratio(v_i) = 7/3$.

Figure 9.4 shows the plot of an experiment in which $n = 10^4$, $\ell = 4$ and $p = 1/\sqrt{n}$. The probability q is set by increasing the ratio $q/p = 2\beta$. Thus, each unique value of q, or equivalently q/p, corresponds to a unique graph with different distribution of intra, inter triangles. For each graph, we record the $Ratio(v)$ on every vertex; and plot it as a *histogram*. Finally, we combine the histograms into a single plot i.e. Figure 9.4. Each colour corresponds to the histogram of a unique value of q/p i.e. *purple* corresponds to $q/p = 2$ i.e. $\beta = 1$; *green* to $q/p = 4$ i.e. $\beta = 2$ and so on.

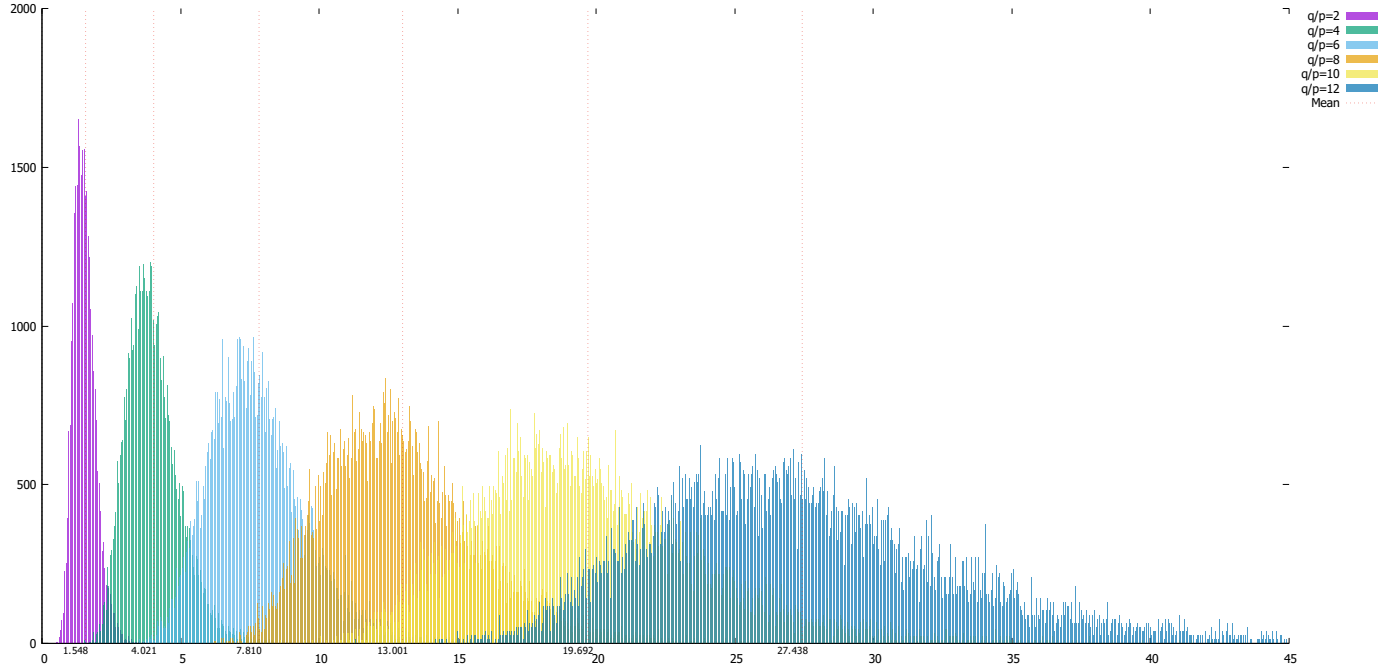


Fig. 9.4 The figure shows the histogram of $Ratio(v)$ as a function of q/p. The graphs are generated with: $n = 10^4$, $\ell = 4$ and $p = 1/\sqrt{n} = 0.01$. Each colour corresponds to the histogram of a unique value of q or equivalently $q/p = 2\beta > 1$ i.e. *purple* corresponds to $q/p = 2$.The *dotted*-lines indicate the means of the distribution of $Ratio(v)$ for each value of q/p. As the ratio q/p is increased, the mean increases and the histograms are flattened.

9.2 Metrics for evaluation of k-MXT on the hidden block model

9.2.1 Fraction of incorrect directed-edges in k-MXT fragments

Assuming the partitions are indexed from $0, 1, \ldots$; let $C(v_i)$ denote the index of the partition that v_i belongs to, then we can get the index of the hidden partition by $C(v_i) = \lfloor i/\ell \rfloor$. We define an *incorrect directed-edge* as an edge selected by the k-MXT algorithm that connects a pair of vertices from different partitions. In other words, if the edge (v,u) is added i.e. u is one of the k selected neighbours of v; and

$$C(v) \neq C(u)$$

then it is an incorrect edge.

Recall that we denote H as a subgraph of G, induced by the directed-edges from the k neighbours selection of each vertex, and thus $V(H) = |V|$ and $E(H) \leq |V| \times k$. We define an index called *fraction of incorrect directed-edge* (FoID), which measures the ratio between the total number of *incorrect* edges over the number of edges in subgraph H. That is, for each directed edge in H, we simply count

$$S(v_i, v_j) = \begin{cases} 1 \text{ if } C(v_i) \neq C(v_j) \\ 0 \text{ otherwise} \end{cases}$$

Then, the index is defined by

$$\text{FoID} = \frac{\sum_{(v_i,v_j)\in E(H)} S(v_i, v_j)}{|E(H)|}$$

Thus, the values of FoID ranges from $[0;1]$ with 0 indicates that every edge in H is *pointing inward* connecting a pair of vertices in the same partition, and a score of 1 being all edges are *pointing* across partitions.

9.2.2 Adjusted Rand Index

As each vertex's final *membership* is decided by its connected components in H, we are interested in comparing the connected components in H with the initial hidden partitions. More specifically, initially, we are given $P = \{P_1, \ldots, P_4\}$ where each P_i is a hidden partition and each consists of $n/4$ number of vertices. Assuming the set of connected components in H is $S(H) = \{C_1, C_2, \ldots, C_m\}$ $(m \geq 1)$ where each C_i is a connected component in H, we would like to know the similarities between $S(H)$ and P.

For that purpose, we use the Adjusted Rand Index (ARI). The measure was introduced in Section 7.3.1 page 124. We provide a short reminder below.

Given a set S consisting of n elements and two partitions derived from S: $X = \{X_1, \ldots, X_m\}$ and $Y = \{Y_1, \ldots Y_n\}$, where m and n may or may not be equal. The ARI measures the similarities of X and Y, yielding a score ranges from $[-1;1]$ with 1 being an exact match i.e. there exists an exact match $X_i = Y_j$ for every i, j and furthermore the cardinality of two sets X and Y are equal i.e. $n = m$.

9.3 Experiment setting and results

In these experiments, we set $\ell = 4$ and conducted two experiments. The objective of the *first* experiment is to investigate the accuracy of the algorithm as a function of $q/p = 2\beta$; in which p *is fixed* at $p = n^{-c} = 1/\sqrt{n}$ thus $c = 1/2$. The objective of the *second* experiment is similar i.e. accuracy as a function of q/p; however with values of p in different thresholds as studied in Section 9.1.3 i.e. $p = n^{-c}$ with $c < 1/2$, $c = 1/2$ and $c > 1/2$. In more details, the experiments are set up as follows

1. Set $n = 10^4$, fix $p = 1/\sqrt{n}$; q is set by increasing $q/p = 0, 1, \ldots, 14$. For each unique set of parameters, we generate 10 different graphs $G(n,p)$ and fragment it with $k = 1,2,3,4,5$ which we abbreviate as $1,2,3,4,5$-MXT. Figure 9.5 shows the results for this experiment.

2. We experiment with different values of p, and hence we need slightly larger graphs. Particularly, we set $n = 4 \times 10^4$ and $\ell = 4$. Thus each partition has size $m = 10^4$. We use three different p values determined by $p = n^{-c}$ with $c = 0.4, 0.5, 0.6$ corresponding to the thresholds of c i.e. $c < 1/2$, $c = 1/2$ and $c > 1/2$. We increase $q/p = 0, 1, \ldots, 10$. For this experiment, we only apply the 5-MXT algorithm to select as many out-edges.

Table 9.1 summarises the parameters used for generating input graphs.

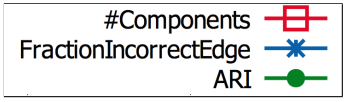
Exp No	n	g	c ($p = 1/n^c$)	q/p	β
1	10,000	2500	0.5	[1,14]	[1,7]
2	40,000	10,000	0.6 0.5 0.4	[1,10]	[1,5]

Table 9.1 Experiments with k-MXT on *hidden block model* with $\ell = 4$.

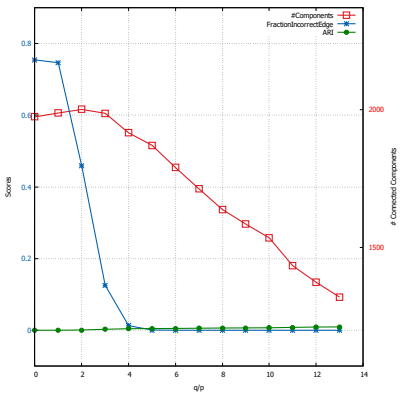
The results are shown in Figure 9.5 and 9.6. A discussion is provided in each figure's caption.

9.4 Conclusion

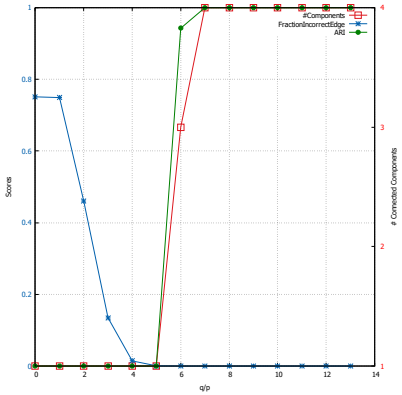
In this chapter, we studied the application of the k-MXT algorithm on the hidden ℓ partition model. We showed that there exists some thresholds of q in relation to p (for some values of p), above which the algorithm is able to recover the correct hidden partitions with $k \geq 2$. This indicates that the k-MXT, as a graph clustering algorithm, works well for graphs with *many triangles*.



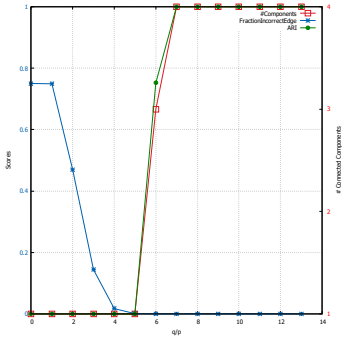
(a) Legends for the figures below



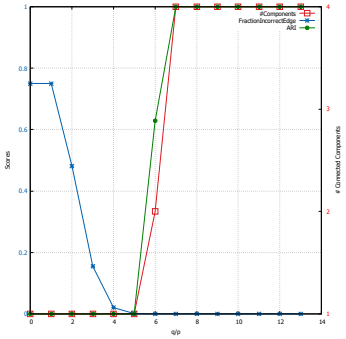
(b) 1-MXT



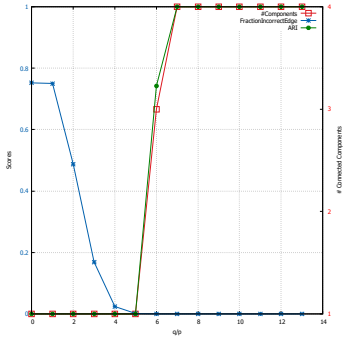
(c) 2-MXT



(d) 3-MXT



(e) 4-MXT

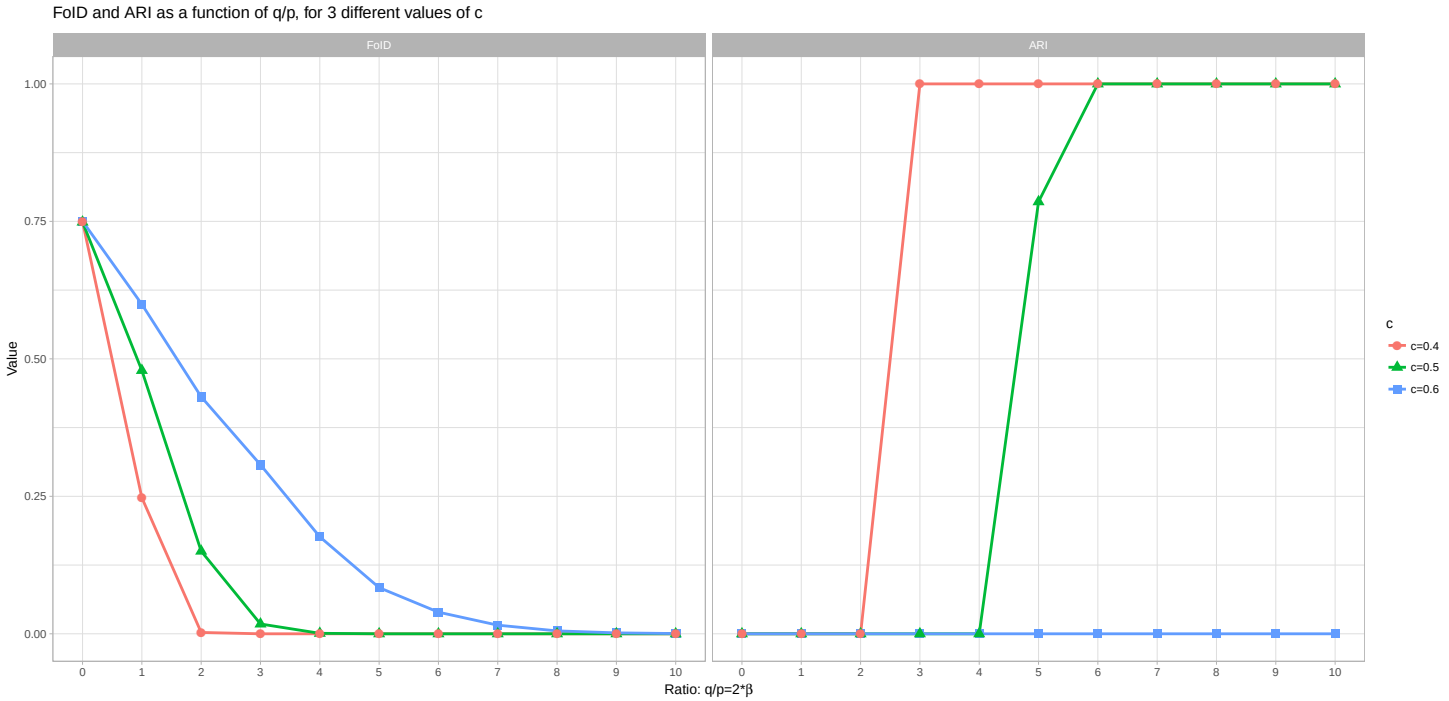


(f) 5-MXT

Fig. 9.5 These figures show a summarised plots of the performance of k-MXT algorithm in the *first experiment*. The **primary** y**-axis** (left) corresponds to the scores for the FoID (blue) and ARI (green). The **secondary** y**-axis** (right) corresponds to the *number of connected components* (red). It can be seen that there is a difference between 1-MXT and the rest, with the former being a distinct case.

That is, *for* 1*-MXT*, although the *FoID* ≈ 0 i.e. the number of *incorrect edges* is 0 (for $q/p \geq 6$) the accuracy measure ARI is also near 0. This is because the 1-MXT *breaks the correct hidden partitions into many small fragments* (as indicated by the red - # components line). Whereas, to achieve a high ARI scores, it is required that the number of components to be close to 4 i.e. the number of hidden partitions. Therefore, even though 1-MXT *does not select any incorrect* (across partition) edge, its accuracy score is near 0. Picturesquely, 1-MXT produces *small fragments* which are contained inside the correct, hidden partitions.

It appears that $2,3,4,5$-MXT *have a threshold at* $q/p \approx 8$, above which the algorithms find the correct partitions i.e. number of components is 4 and ARI scores are 1.



(a) 5-MXT

Fig. 9.6 The figures plot the Fraction of Incorrect Edge - FoID *(left figure)* and Adjusted Rand Index - ARI (right figure). The measures are plotted as a function of the ratio $q/p = 2\beta$ for three different values of c i.e. $p = n^{-c}$: $c = 0.4$ *red*; $c = 0.5$ *green* and $c = 0.6$ *blue*.

These values of c corresponds to the threshold of p as analysed in Section 9.7. That is, we established that for $c > 1/2$ i.e. $c = 0.6$, then there is an insufficient number of triangles, thus β needs to be large in order to find the correct partitions. Else, for $c \leq 1/2$ there exists thresholds, above which the k-MXT should be able to find the correct partitions.

It is seen that with $c = 0.6$, although the FoID seemingly approaches 0 however the ARI remains approximately 0. This indicates that there are some *incorrect edges* selected. Since the algorithm is very sensitive to *cross-partition edges* i.e. an incorrect edge would connect two partitions into a single connected component, the ARI would be diminished. On the other hand, with $c == 1/2$ and $c = 0.4 < 1/2$, the algorithm reaches the ARI score of 1 (correct results) much earlier. The sharp rises in both functions indicate the thresholds.

Chapter 10

Application of k-MXT to graph clustering for real world networks

In this chapter, we study the results produced by the k-MXT algorithm on a set of graphs consisting of real world networks, in the sense that each vertex represents an *object* in real life e.g. human, animal, etc. and the edges represent the interactions between them. Also some artificial networks that are generated by mathematical models.

These graphs share a common property, that the vertices have the tendency to divide into groups such that vertices in each group are more tightly connected, compared to the rest. This feature can be regarded as the *community structure* of a network. The task of identifying the communities if they exist in the underlying graph is known as the *graph clustering* or *community detection* problem. Chapter 7 provides a literature review on the problem.

The fundamental idea of *triangles*, is a foundation of the *clustering coefficient* which is a common characteristic of communities found in many real world networks. Recall that the k-MXT algorithm produces *fragments*. These are connected components formed by mapping every vertex to its k adjacent vertices which have the highest number of common neighbours i.e. triangle-fragments. This motivates us to study the application of k-MXT on networks with community structure.

In this chapter, we study the application of k-MXT to graphs with community structures. More particularly, we examine the *output fragments* of these graphs and ask whether such fragments can be considered *communities* or parts of communities. In order to do so, we need a set of evaluation tools. Firstly, we need a set of networks which have apparent community structures and the a-priori communities are known. For this, we select some networks which were introduced in Section 7.2 (page 117). Secondly, we need a criteria to measure how similar are the apriori communities and the output; and the *goodness* of the fragments. We used some measures that were discussed in Section 7.3 (page 123) to compare the results with some popular community detection algorithms (see Section 7.4 page 127).

This chapter is structured as follows. Section 10.1 and Section 10.2 briefly review the testing networks and evaluation criteria, respectively. Section 10.3 explains the setting of the experiments. In

Section 10.4 and 10.5, we present the experimental results and discuss a modification to the original k-MXT algorithm.

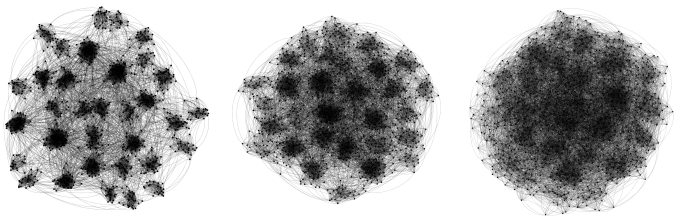
By its nature k-MXT can only be used on networks with an adequately large clustering coefficient. Furthermore, we only consider the case where each vertex belongs to *only one* (non-overlapping) community i.e. *hard clustering*. This is because, for this current version of the algorithm, each vertex's *community* is determined by its connected component; and each vertex belongs to exactly one connected component.

10.1 Testing networks

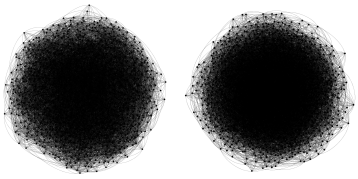
We use both real-world and computer-generated networks as testing graphs. For the real world networks, we use the following:

1. the Karate club,
2. the Bottle-nose dolphins network,
3. the political books network,
4. the football clubs network.

For computed-generated network, we use the Lancichinetti-Fortunato-Radicchi (LFR) benchmark graph. For more detailed description of these networks, see Section 7.2 on page 117. Below we provide the visualisations of these networks.

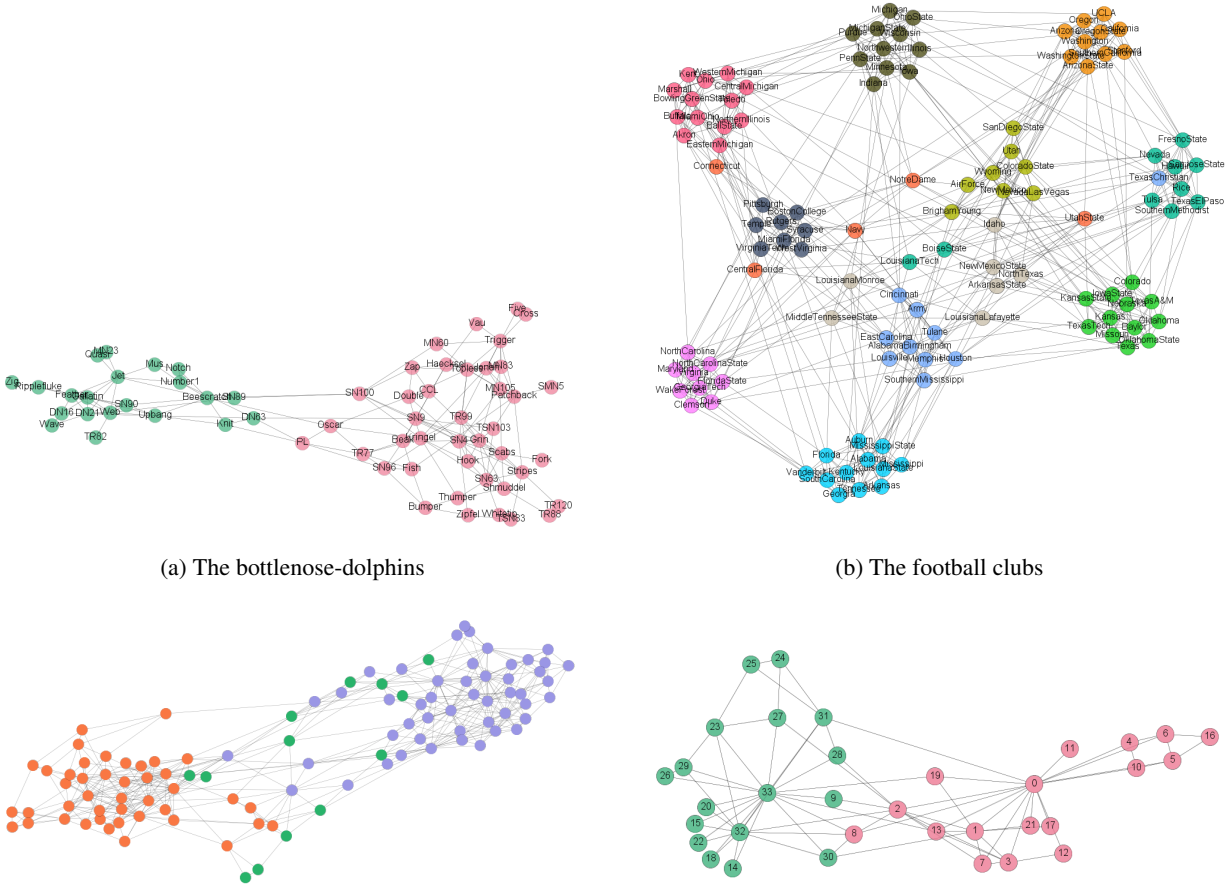


(a) $\mu = 0.1$ (b) $\mu = 0.2$ (c) $\mu = 0.3$



(d) $\mu = 0.5$ (e) $\mu = 1.0$

Fig. 10.1 A visualisation of the LFR-benchmark graph with $n = 1,000$. The cluster sizes vary in range $[20-50]$ and the value of the *mixing-parameter* μ increases from 0.1 to 1.0. It is seen that the cluster-structure of the graph gradually disappears as μ increases.



(a) The bottlenose-dolphins

(b) The football clubs

(c) The political books network

(d) The karate club

Fig. 10.2 The input social networks. For each vertex in each network, its colour corresponds to its ground-truth community. (a) The network bottlenose-dolphins, the two communities are quite easily identifiable given there are very few edges joining the two. (b) The football teams, the colour represents each team's conference. There are teams appear to be more closely connected to teams in another conference. This is due to the games are geographically dependant. (c) Books in *orange* align with *liberal*; *purple* with *conservative* and *green* are *neutral*. (d) Vertex 0 is *the instructor*, vertex 33 is *the administrator*. Visually, the centrality of these two vertices are apparent. One can also roughly identify the separation of such network, with some difficulties at vertices $13, 2, 8$ which lie between the two groups.

10.2 Evaluation/Comparison

In this section, we introduce the criteria for evaluating the performance, particularly the accuracy, of the algorithms. Recall that each input graph has a *ground truth* community structure [44] which can be considered the *answer* to each problem. Hence, we need to derive a criterion to establish how *similar* the output is to these apriori communities.

Consider the ground-truth communities. As each vertex belongs to *exactly* one community, let the truth communities be the set $\mathscr{T} = \{T_1, T_2 \dots T_i\}$, where each T_i is a *disjoint* set of vertices e.g. $T_1 \cap T_2 = \emptyset$; and hence $T_1 \cup \dots \cup T_i = V$. Next, let the set of output communities be $\mathscr{C} = \{C_1, C_2 \dots C_j\}$, where j may or may not equal i; and each set $C_j = \{v, w, \dots\}$ is a *disjoint* set of vertices $C_1 \cup \dots \cup C_j = V$. Then, we would like a function $f(\mathscr{T}, \mathscr{C}) \to x \in R$ that evaluates the similarity x between the two sets. The Adjusted Rand Index (ARI) (see Section 7.3.1 page 124) seems to be the perfect as per the problem description and hence is our method of choice.

10.3 Experiment setting

For comparison with the k-MXT, we use the community detection algorithms which were introduced in Section 7.4 (page 127). The algorithms are abbreviated as follows

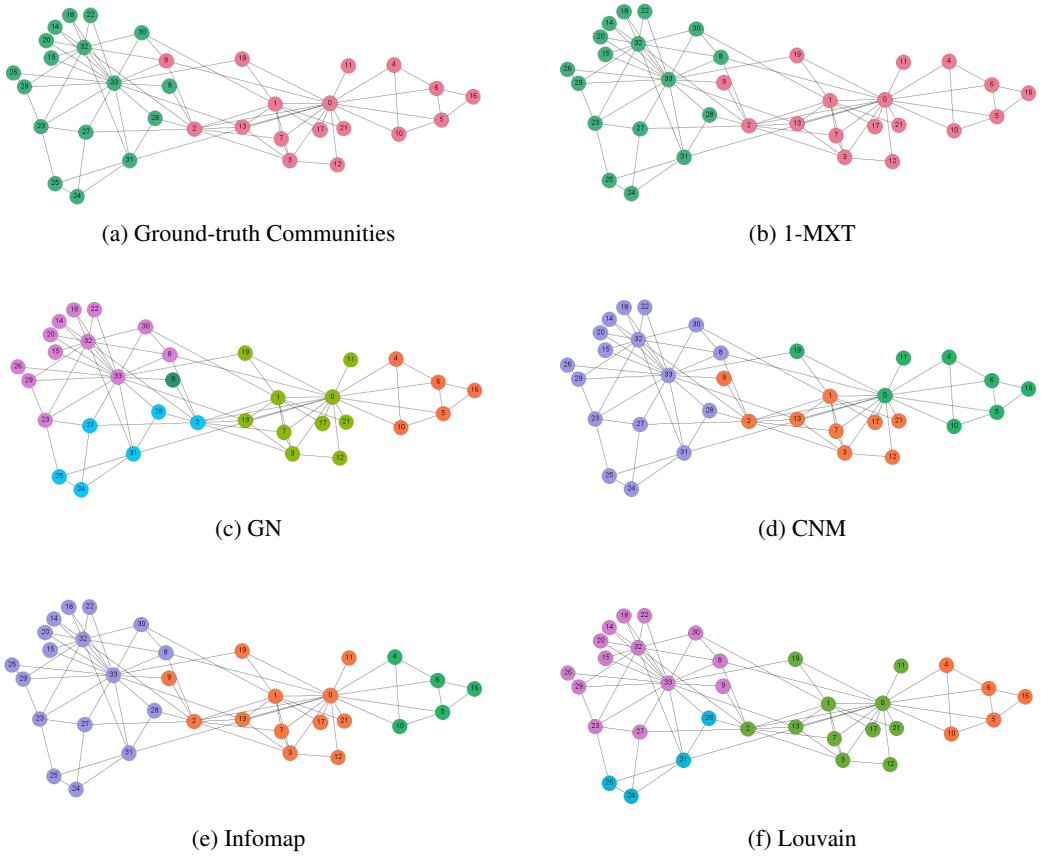
- Triangles fragmentation as k-MXT, we include 1-**MXT** and 2-**MXT**;
- Edge-betweenness clustering as **GN**;
- Fast modularity optimisation as **CNM**;
- Fast greedy modularity as **Louvain**;
- Information encoding as **Infomap**.

Regarding implementation, the GN and CNM algorithms are executed using the Stanford Network Analysis Platform (SNAP) library [35]. The Infomap and Louvain are executed using the software package provided by the authors, available at [1] and [3] respectively. All algorithms are run on a standard laptop.

10.4 Experimental results

We present the visualisations of the results below.

10.4.1 Communities of the Karate network



(a) Ground-truth Communities

(b) 1-MXT

(c) GN

(d) CNM

(e) Infomap

(f) Louvain

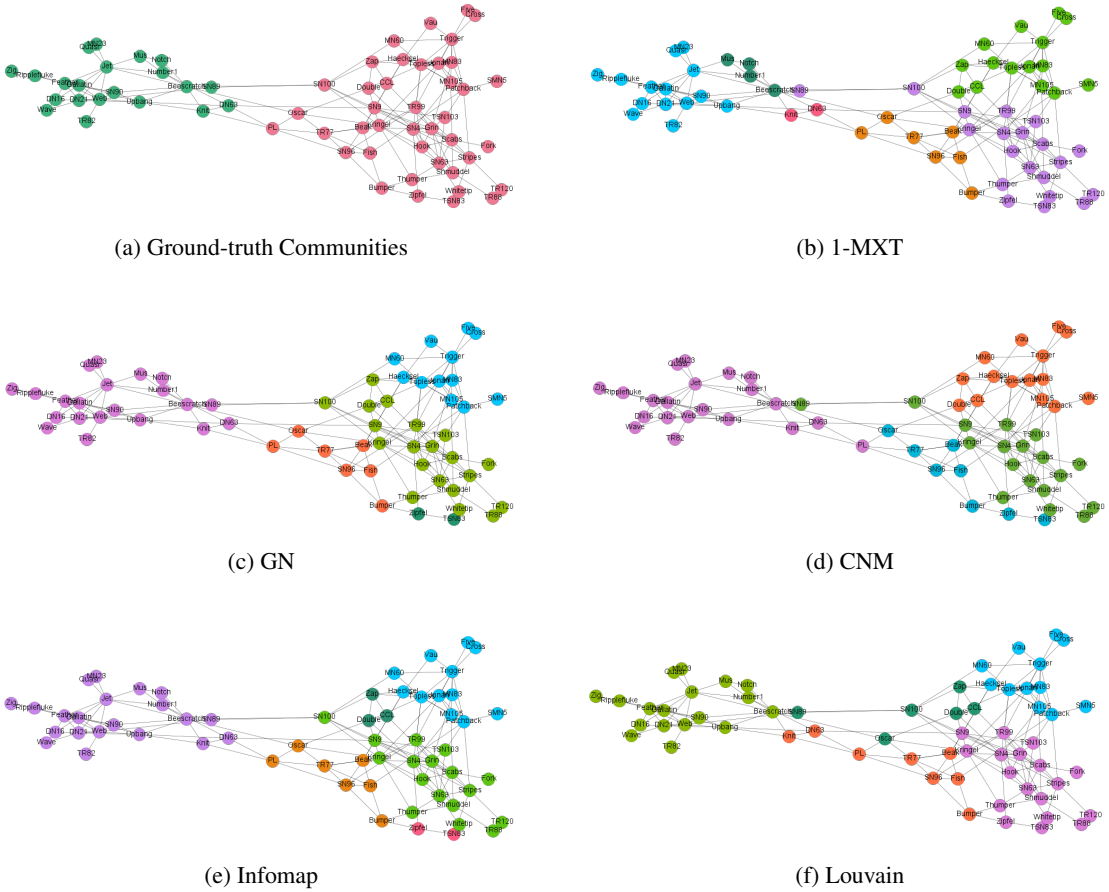
Fig. 10.3 The above figures show the communities of the *Karate network*. Figure (a) shows the *ground-truth community structure* of the network. Figure (b) - (f) show the communities produced by the algorithms, which are labelled accordingly. Visually, the result of 1-MXT closely match the ground-truth communities, the two mismatches occur at vertex 8 and 9. The remaining algorithms produce more communities than in the ground-truth. The 2-MXT visualisation is excluded above because it returns a single cluster.

Table 10.1 presents the ARI scores and modularity for the communities produced by each algorithm. It is seen that the 1-MXT produces the highest score. Hence is the most similar cluster compares to the ground-truth communities, which agrees with the visual inspection. In terms of modularity, Louvain algorithm produces the best clusters.

	1-MXT	2-MXT	GN	CNM	Infomap	Louvain
ARI	**0.742**	0.000	0.392	0.568	0.590	0.508
Modularity	0.368	0.000	0.400	0.384	0.400	**0.415**

Table 10.1 Results of clustering algorithms on the *karate* network.

10.4.2 Communities of the Dolphins network



(a) Ground-truth Communities

(b) 1-MXT

(c) GN

(d) CNM

(e) Infomap

(f) Louvain

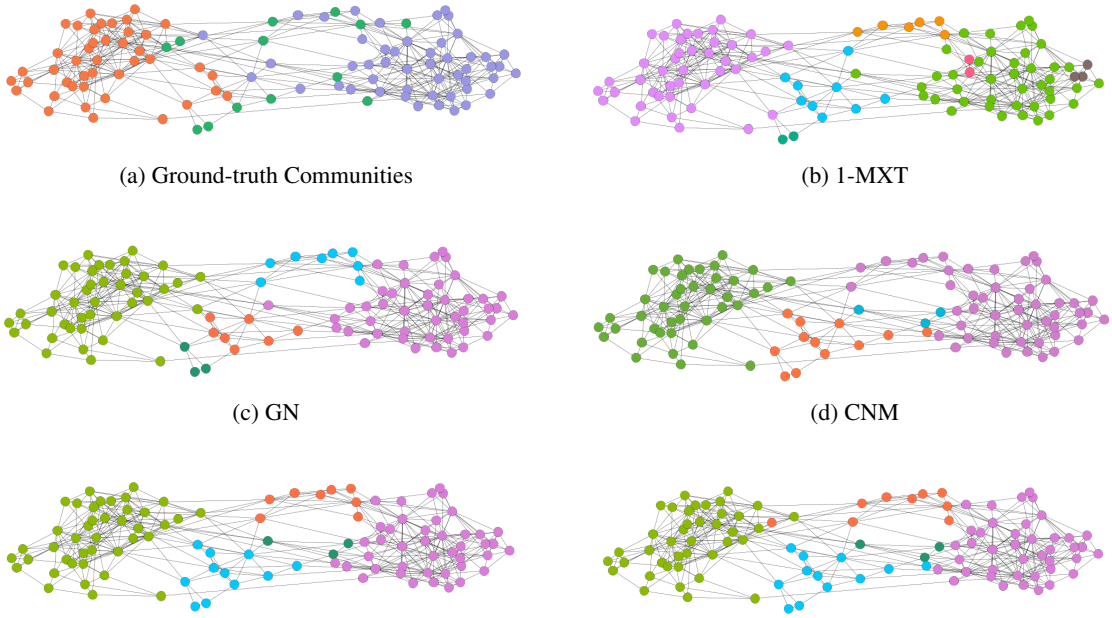
Fig. 10.4 The above figures show the communities of the *Dolphins network*. Figure (a) shows the *ground-truth community structure* of the network. Figure (b) - (f) show the communities produced by the algorithms, which are labelled accordingly. Overall, all algorithms produce more communities than the ground-truth. Visually, it is difficult to judge which is the best result. Nevertheless, GN and Infomap are arguably the closet to resemble the ground truth. The reason is that these algorithms produce a relatively similar *split* as in the ground-truth.

Table 10.2 presents the ARI scores and modularity for the communities produced by each algorithm. The best algorithms according to the ARI and Q score are GN and Infomap, respectively.

	1-MXT	2-MXT	GN	CNM	Infomap	Louvain
ARI	0.341	-0.010	**0.451**	0.383	0.401	0.346
Modularity	0.496	0.257	0.519	0.513	**0.523**	0.518

Table 10.2 Results on the *Dolphins* network.

10.4.3 Communities of the Polbook network



(a) Ground-truth Communities (b) 1-MXT

(c) GN (d) CNM

(e) Infomap (f) Louvain

Fig. 10.5 The above figures show the communities of the *Political books network*. Figure (a) shows the *ground-truth community structure* of the network. In the ground-truth network: the *orange* vertices represent books align with *liberal*; and *light purple* with *conservative*. The *green* vertices are *neutral* books. Notably, the ground-truth communities appear to be not really well-separated.

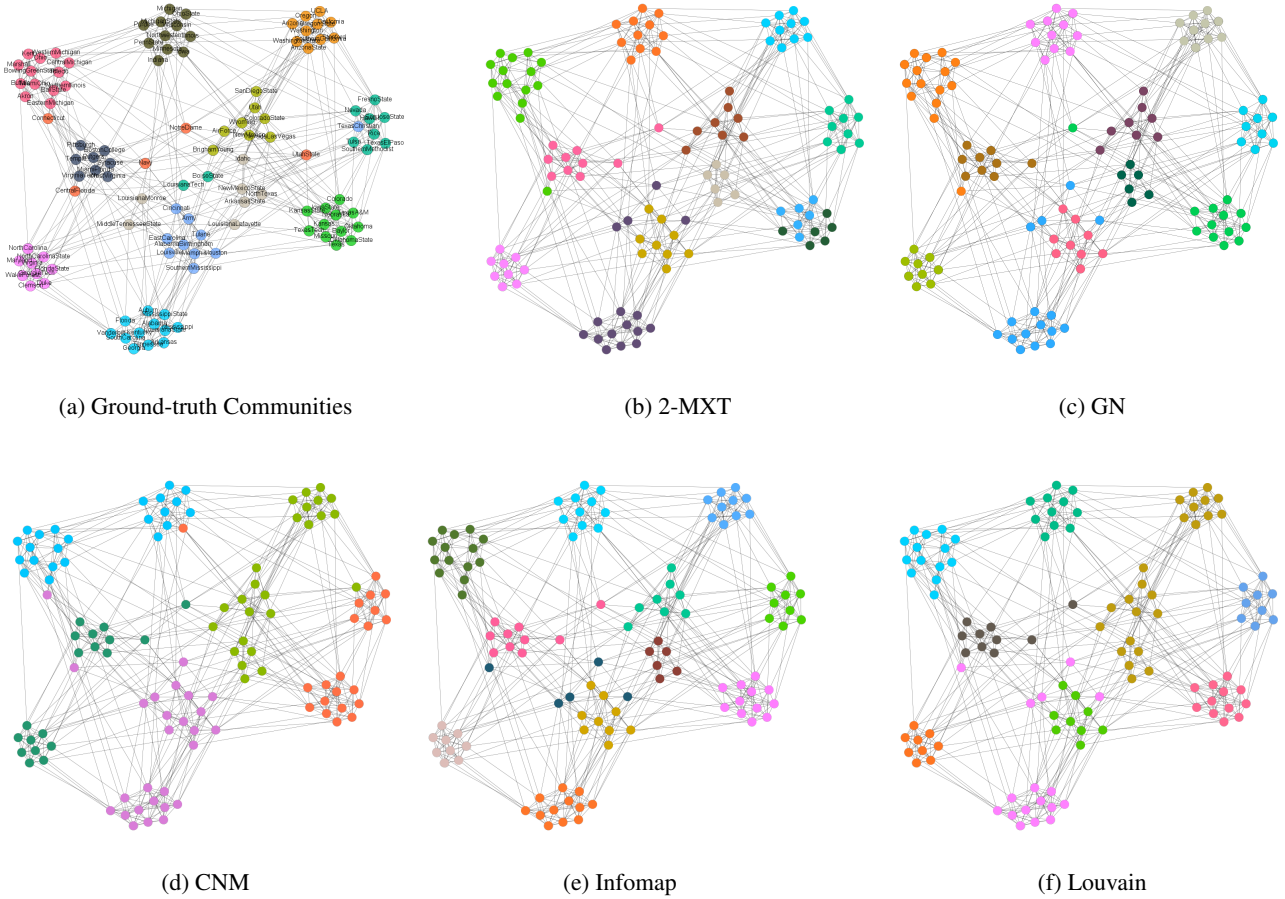
Visually, the 1-MXT communities are quite similar to GN's. However, a major difference is that 1-MXT produces two small clusters i.e. the *light-red* (2 vertices) and *brown* (3 vertices) clusters, which are *contained* inside the large community consists of green vertices.

According to Table 10.3, GN and Infomap are the best algorithms in terms of ARI and modularity, respectively.

	1-MXT	2-MXT	GN	CNM	Infomap	Louvain
ARI	0.511	0.000	**0.682**	0.638	0.646	0.642
Modularity	0.485	0.000	0.515	0.502	**0.526**	0.520

Table 10.3 Results on the *Political books network*.

10.4.4 Communities of the Football network



(a) Ground-truth Communities (b) 2-MXT (c) GN

(d) CNM (e) Infomap (f) Louvain

Fig. 10.6 For the *football network* we show the result of 2-MXT which is better compares to 1-MXT. For the detailed scores of the algorithms, see Table 10.4.

Summarised result on real-world networks.

(a) ARI

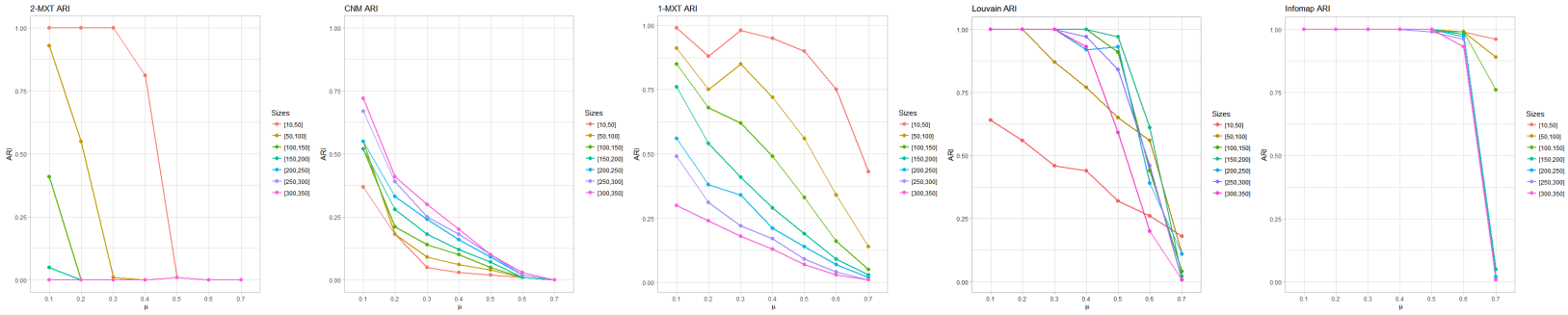
	1-MXT	2-MXT	GN	CNM	Infomap	Louvain
Karate	**0.742**	0.000	0.392	0.568	0.590	0.508
Dolphins	0.341	-0.010	**0.451**	0.383	0.401	0.346
Polbook	0.511	0.000	**0.682**	0.638	0.646	0.642
Football	0.454	0.805	0.842	0.447	**0.890**	0.704

(b) Q

	1-MXT	2-MXT	GN	CNM	Infomap	Louvain
Karate	0.368	0.000	0.400	0.384	0.400	**0.415**
Dolphins	0.496	0.257	0.519	0.513	**0.523**	0.518
Polbook	0.485	0.000	0.515	0.502	**0.526**	0.520
Football	0.279	0.571	**0.601**	0.566	0.600	0.600

Table 10.4 Performance of the algorithms on the social/real networks introduced in Section 10.1. Table (a) presents the Adjusted Rand Index scores of the algorithms. Table (b) presents the modularity of the result communities. The highest scores are highlighted. Note that 1-MXT performs well on the Karate network and 2-MXT performs well on the Football network, as noted in previous figures.

Results on LFR benchmark graphs.



(a) 2-MXT (b) CNM (c) 1-MXT (d) Louvain (e) Infomap

Fig. 10.7 Performance of the algorithms on the LFR graphs. The plots show the ARI scores as a function of the mixing parameter μ. The different curves correspond to different community size ranges e.g. $[10,50]$ (red), $[50,100]$ (brown), etc. Due to its high complexity, the GN algorithm was excluded in this experiment. The algorithms are sorted according to the overall performance from worst (left) to best (right).

10.4.5 Accuracy

Table 10.4 presents results for the *real-world networks*. Figure 10.7 shows the ARI results on the *LFR benchmark graphs*. From the results of all tests, Infomap appears to be the best algorithm; follows by Louvain, then 1-MXT/CNM/GN and finally 2-MXT. The recorded performance of the comapred algorithms i.e. Infomap, Louvain agrees with the findings in [20].

Interestingly, 2-MXT performs very well on the football network and also on LFR but only for specific parameters i.e. Figure 10.7 (a) red curve. Furthermore, on the LFR tests, 1-MXT appears to produce better results compares to CNM. However, its performance seems to be dependent on the community sizes parameter, which determines the number of triangles (hence clustering coefficient) of the input graph.

10.4.6 Experimental running time

A detailed complexity analysis of the k-MXT algorithm is discussed in Section 12.6.2.2. Here we provide the experimental running time of the tested algorithms.

The real-world networks are small. Hence to test the complexity/running-time, we need larger graphs as input. We used two graph models: LFR benchmark and $G(n,p)$. Using the LFR graphs, we can control *the community sizes* and *maximum degree* of the input graphs. The random graph model is considered as a neutral test i.e. the input graph has no community. Using $G(n,p)$, we can control the number of *vertices* and *edges*. In the experiments, we observed the effect of these parameters i.e. *community sizes*, number of *vertices, edges* and *maximum degree*, on the running time of the algorithms.

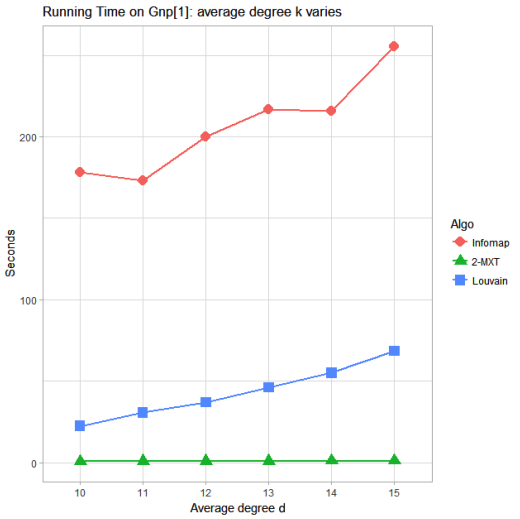
Table 10.5 summarises the parameters of the input graphs. For each test, we plot the *average running time* of each algorithm over *10 executions*, as the function of the varied parameters.

Overall, k-MXT appears to have a good running time in cases (a) (b) (c), except for (d) in which $d_{\max}$ varies. k-MXT running time is dependant on the maximum degree of the graph i.e. d_{max}, which is apparent from Figure 10.5-(d) (as analysed in Section 12.6.2.2).

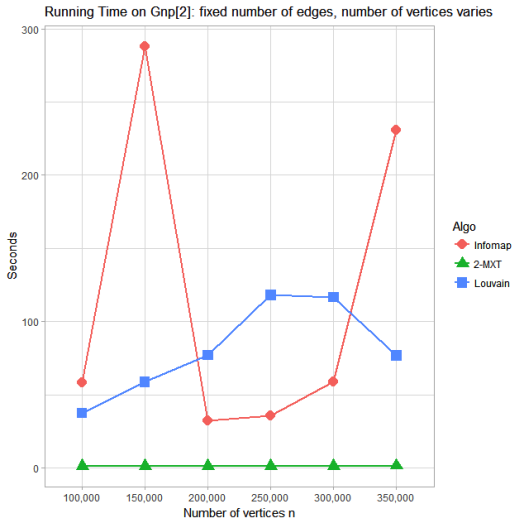
Louvain and Infomap have much better running time on LFR graphs than on $G(n,p)$. These algorithms do not work very well when the underlying graph does not have a clear community-structure. A possible reason is that Louvain works by merging the pair vertices which yield the highest increase in modularity. Since the input has low modularity, such optimisation strategy would be inefficient.

Infomap seems to have high variable running times across the tests. On the contrary, Louvain's running time appears to be consistent.

Also, it is worth noting that both Louvain and Infomap perform significant better when d_{max} increases i.e. Figure 10.8(d). In previous tests i.e. LFR1 and LFR2, the graphs are more *regular* i.e. the degree distribution is uniform, this is controlled by setting the maximum degree equals to the average degree. In LFR3, as the max degree parameter is increased, the running time of Louvain is slightly improved while Infomap's is significantly reduced.

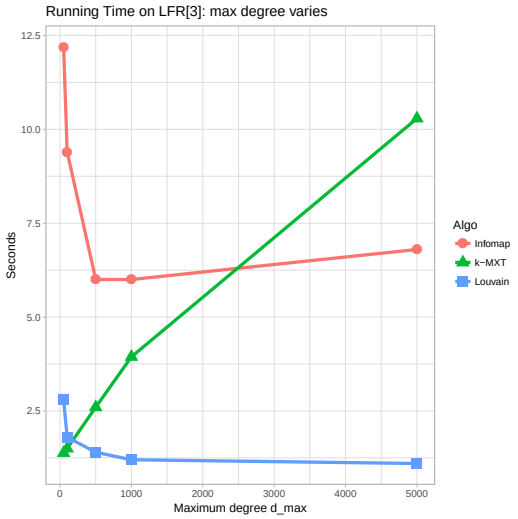


(a) Gnp1



(b) Gnp2

(c) LFR1, LFR2



(d) LFR3

Fig. 10.8 Running time is recorded in *seconds* and is averaged from the results of 10 executions. Note that we used $k = 2$ for the k-MXT algorithm i.e. 2-MXT.

In figure (a) we generate random graphs with *number of vertices fixed*. The plot shows the running time as a function of average degree k, for $k = 10, \ldots, 15$.

In figure (b) we generate random graphs with *average degree fixed*. The plot shows the running time as a function of the number vertices n, for $n = 10^5, \ldots, 3.5 \times 10^5$.

In figure (c) we generate two LFR graphs. The first graph, LFR1, with *smaller community size gap*; whereas for the second graph, LFR2, we use a *larger size gap* e.g. $[10, 10^3]$ and $[10, 10^4]$. The **red** (a) curves show results on the LFR1. The **blue** (b) curves show results on the LFR2. In both cases, the Louvain and k-MXT running times are relatively fast; whereas the Infomap has apparent slower running time.

In figure (d) we generate LFR graphs with *maximum degree* increases i.e. $d_{\max} = 50, 100, 500, 1000, 5000$. We attempted to generate input graphs with $d_{\max} = 2000, 3000, 4000$, however the graph generator did not return results.

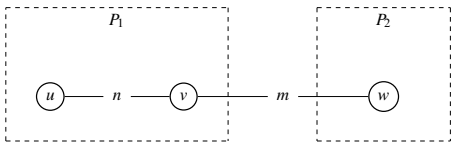
Graph	Model	Parameter		Remarks	Plot
		Vertices	Edges		
Gnp1	$G(n,p)$	100,000	Varies	Fixed number of vertices $\|E\| = 10^6, \ldots, 1.5 \times 10^6$	10.8(a)
Gnp2	$G(n,p)$	Varies	750,000	Fixed number of edges; $\|V\| = 10^5, \ldots, 3.5 \times 10^5$	10.8(b)
LFR1	LFR	100,000	1,500,000	Max degree, average degree = 15. Size varies, small gap: $[minc, maxc] = [10, 10^3], [10^3, 5 \times 10^3], etc.$	10.8(c)-red
LFR2	LFR	100,000	1,500,000	Max degree, average degree = 15. Size varies, large gap: $[minc, maxc] = [10, 10^4], [10^4, 2 \times 10^4], etc.$	10.8(c)-blue
LFR3	LFR	100,000	1,500,000	Fixed size range: $[10, 1000]$. Average degree = 15. Max degree varies: $maxD = \{50, 100, 500, 1000, 5000\}$	10.8(d)

Table 10.5 The graphs used in the complexity tests.

10.5 Parameterised k-MXT $\left(k\text{-MXT}(w) \right)$

From the experiments conducted in previous sections, we draw some conclusions on the practical implications of the k-MXT algorithm. Firstly, it has a relatively good running time if the input graph is sparse. Secondly, the empirical results are reasonable, however not on par with some of the current best algorithms (DBSCAN [30]). Furthermore, an apparent weakness is that the quality of the partitions worsen significantly as k increases. *So how can we improve the performance of k-MXT?*

Consider the following scenario. Suppose there is a vertex v which belongs in the ground-truth community P_1. Next, v is connected to $u \in P_1$ and $y \in P_2$. Denote a vertex which has connections to other partitions as a *boundary* vertex. Suppose the weights of the edge (u,v) and (v,w) are n and m, respectively. Further, for v: n is the highest and m is the second highest edge weight.



In this scenario, if 1-MXT is applied, then v selects u. If $k = 2$, then v selects both. Then, the likely case is that the communities or components P_1 and P_2 would be connected, which is a less desired outcome. This is the weakness of the algorithm i.e. the k-MXT is very sensitive to *incorrect edges*. Similar cases occur very often in practice. In these scenarios, the *boundary* vertices usually

have high weight *inter-edges*. Thus, by increasing k, the algorithm is more prone to error by choosing the inter-edges thus joining communities together.

In the following sections, we study a parameterised version of the k-MXT: k-MXT(w) where w is an additional parameter called *minimum weight*. The w parameter works as an *edge filter*: for every vertex v, we only consider the incident edge which has larger weight than w. Thus, equivalently, the number of edges in the graph G decreases with w; in other words, G gets sparser with w. Consequently, if w is set to a correct value i.e. $w > m$ in previous example, then such edge would be *filtered out*, leaving only *correct* edge.

The introduction of w would create the following scenario. Consider a vertex v and its set of incident edges $E(v)$. If for every edge $e \in E(v)$ $w(e) < w$, then there is no out-edge from v. In such case, we denote v as an *isolated* vertex or a *noise* vertex.

The notion of *noise* seems to be absent in the field of graph clustering. Consequently, we lack a set of tools for evaluating results in this category; which is problematic for evaluating the performance of k-MXT(w). To compensate for this, we consider some modifications to the introduced measures as follows.

10.5.1 Revised ARI

Let $G(V,E)$ be the input graph. For each vertex $v \in V$, let $N(v)$ be its adjacency list. For every edge $e(v,u)$, the edge is given a weight

$$w\{e(v,u)\} = |N(v) \cap N(u)|.$$

Set the minimum weight parameter $\overline{w}$. Suppose for a vertex v of which every incident edge has weight less than $\overline{w}$ i.e.

$$w\{e(v,u)\} < \overline{w} : \forall u \in N(v),$$

then v is an *isolated vertex*. At the end of k-MXT, every isolated vertex is a singleton and we can choose whether to consider it as a community. Generally, we do not consider isolated vertex as community and remove it from the result.

Now, let $\mathscr{T}$ be the set of the ground truth communities, and let I be the set of isolated vertices. We then remove every isolated vertex from the ground truth communities i.e. $T_i \setminus \{v\} : v \in I; T_i \in \mathscr{T}$. We then calculate the ARI measure using the result $\mathscr{T}'$ and the result communities.

In this sense, we are measuring how similar the resulting partitions is to a *subset* of the ground truth communities. Recall that ARI yields a score from $[-1,1]$ where a high scores now indicate the output partitions supposedly lie within the ground truth communities.

10.5.2 Measuring connectivity of the partitions

In previous section, we used the modularity to measure the connectivity/goodness of the resulting partitions. Although there is no concept of noise in modularity, the measure can still be computed in

the absence of isolated vertices since these can be considered a singleton community. Particularly, for each isolated vertex $v \in \mathscr{I}$, the modularity Q_v is then, using equation (7.3),

$$Q_v = -\Big(\frac{d(v)}{2m}\Big)^2,$$

where $d(v)$ is the degree of vertex v, and $2m$ is twice the number of edges in G. Then, for all isolated vertices we get $Q(I) = \sum_{v \in I} Q_v$. Thus, the overall modularity of the result partitions might be undermined if many isolated vertices are included.

Alternatively, we consider measuring the connectivity of the fragments using the *clustering coefficient*, which is also based on the concept of *triangles*.

Suppose the k-MXT(w) produces a partition $\mathscr{P} = \{P_1, \ldots, P_i\}$ (where each P_i is a disjoint set of vertices), and a set of isolated vertices I. Hence the union of these two sets gives the complete vertex set $I \cup \mathscr{P} = V$.

Let $G[P_i]$ be the graph induced by the vertex set P_i. That is, the graph $G(P_i)$ is formed by the vertex set P_i and edges from $G(E)$ which have both ends in P_i. We call these *community-induced subgraphs*. For each subgraph, we compute the clustering coefficient of $G(P_i)$ as given in equation (7.4). Then we can take the *average clustering coefficient*

$$C(G[\mathscr{P}]) = \frac{1}{|\mathscr{P}|} \sum_{P_i \in \mathscr{P}} C(G[P_i]).$$

Thus, the $C(G[\mathscr{P}])$ (or C for simplicity) now measures the *averaged triangle density* in the subgraphs induced by the set of the partitions. In other words, it determines how close these subgraphs are to being a complete graph; which can be an indication of high connectivity among the partitions.

10.5.3 Experiments

In this section, we measure the performance of k-MXT(w). First, we introduce the set of test graphs.

10.5.3.1 Input graphs

We re-use some of the graphs that were introduced in previous section. To grasp the effect of w, the input graphs should have sufficient number of *triangles*. Therefore, we choose the *Football* network and the LFR graph model. To make the test more diverse, we generate two LFR graphs, one with more triangles and one with fewer. More specifically, as pointed out, the triangle density in LFR graph appears to be dependant on number of edges or cluster sizes. Thus, *to increase the number of triangles* in LFR graphs, we choose to *add more edges* and keep the remaining parameters constant, see Table 10.6 for details.

Additionally, we include a real-world *geographical graph*. In this graph, each vertex has a geographical coordinates i.e latitude and longitude. We then connect every pair of vertices which has distance at most $50m$ (see Section 12.6.2.1, Chapter 12 for more details). This dataset consists of

Graph	$\lvert V\rvert$	$\lvert E\rvert$	Parameters	Clustering coeffcient
Football	115	≈ 600	Not Applicable	0.41
LFR1	10,000	$\approx 75,000$	$k = 15, maxD = 50$ $\mu = 0.1, size = [10, 1000]$	0.06
LFR2	10,000	$\approx 250,000$	$k = 50, maxD = 100$ $\mu = 0.1, size = [10, 1000]$	0.12
Geo	3,400	$\approx 250,000$	$D = 50m$	0.72

Table 10.6 Input graphs.

approximately $3,400$ vertices and the maximum distance is set at 50 metres. The resulting graph then has $|E| \approx 250,000$ and contains a large amount of triangles, significantly higher than LFR even with much fewer vertices (see clustering coefficient Table 10.6).

Figure 10.9 plots the density of the *edge weight* (recall that for an edge $e(v,u)$ its weight $w(e) = |Adj(v) \cap Adj(u)|$ where $Adj(v)$ is the adjacency list of v). Thus, the weight of an edge is the number of common neighbours of its incident vertices. It is seen that the LFR1 graph has very sparse edge weight, with the mean edge weight is approximately 1, whereas the Geo graph is very dense.

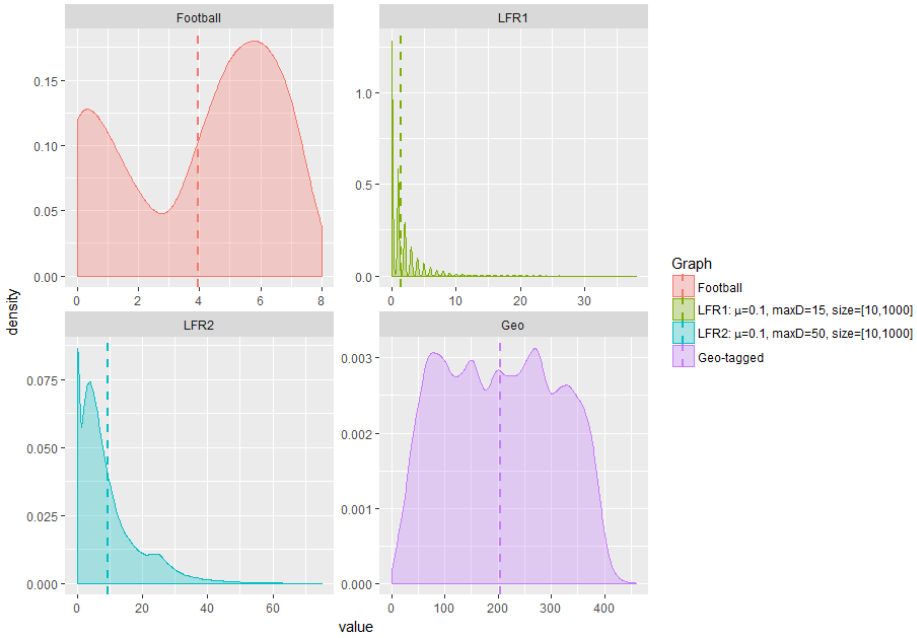


Fig. 10.9 Density plots of edge weight of the input graphs i.e. from left to right, top row: Football, LFR1; bottom row: LFR2, Geo. In each plot, the *dashed line* indicates the *average edge weight*.

10.5.3.2 Testing

There are two influential parameters of the algorithm, k and w. In these experiments, we wish to explore the effect of both parameters. And hence, we conduct the experiments as follows.

To grasp w, we set $k = 2$ (because $k = 1$ creates small, uni-cyclic fragments which is less useful in this case) and increase $w = 0, \ldots, 11$ for Football and LFR; $w = 0, 20, 40, \ldots, 220$ for Geo graph. The results for this experiment are presented in Figure 10.11. To grasp k, we fix w close to the graph's *mean edge-weight*, to leave a sufficient number of edges in the underlying graph; and increase $k = 1, \ldots, 5$. The results for this experiment are presented in Figure 10.12. Note that we ignore communities whose size is less than 3.

For each output, we recorded *six measurements* which are divided into two types as follows:

- Measures which have value range $[0, 1]$:
    1. Revised ARI (except for the geo graph);
    2. Clustering coefficient of the induced subgraphs;
    3. Fraction of isolated vertices;
- Measures regarding the structure of the community-induced subgraph:
    4. Average size of communities;
    5. Number of communities;
    6. Average diameter of the induced subgraphs.

Measures (1) and (2) give an indication of the *goodness* of the result partitions. Additionally, we keep track of the *fraction of isolated vertices* (3). This is because if there are very few remaining vertices, then the resulting clustering coefficient would be less significant. The remaining criteria: (4), (5) and (6) provide some insights of the community-induced subgraphs.

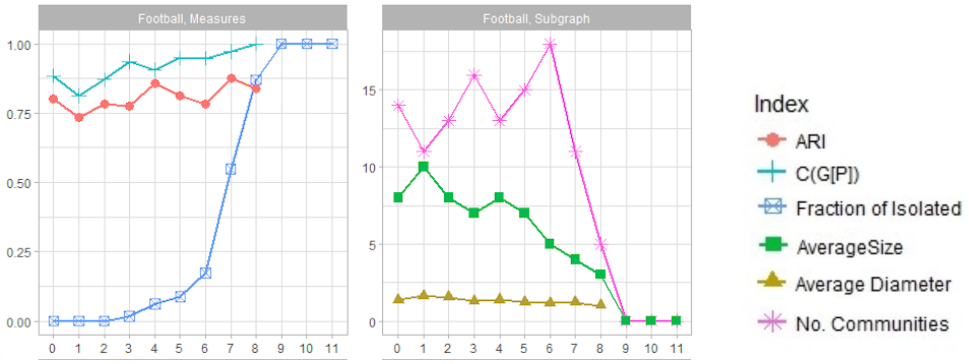
10.5.3.3 Results

Figure 10.11 and Figure 10.12 present the experimental results. In these figures, for each graph we produce *a pair of plots*: one for measures 1,2,3 and the other for measures 4,5,6. The plots are shown in together in pair. For instances, in Figure 10.11 from left to right, on top row we have the pair for: Football and LFR1; and bottom row: LFR2 and Geo. A sample interpretation of the result is given in Figure 10.10.

Figure 10.11 plots performance of the 2-MXT(w) as a function of w. It is seen that as w increases, the more vertices are isolated. The number of isolated vertices increases sharply when w approaches the mean weight. Furthermore, we see that

1. The community-induced subgraphs appear to have have relatively high average clustering coefficient which increases slowly with w.

2. The high ARI scores indicate these most of the fragments lie inside its ground truth communities.



(a) Plots for Football network (b) Legends

Fig. 10.10 The results for k-MXT(w) as a function of w on the **Football** network. The pair of plots is extracted from Figure 10.11. The results are interpreted as follows. The left figure shows the scores for: *ARI, C, Fraction of isolated vertices*. For $w = 1,\ldots 6$, a small fraction of vertices *are isolated*. Furthermore, the resulting fragments have good ARI scores and high clustering coefficient C. For $w \geq 9$, all vertices are isolated. Since they are not considered community, all results are 0.

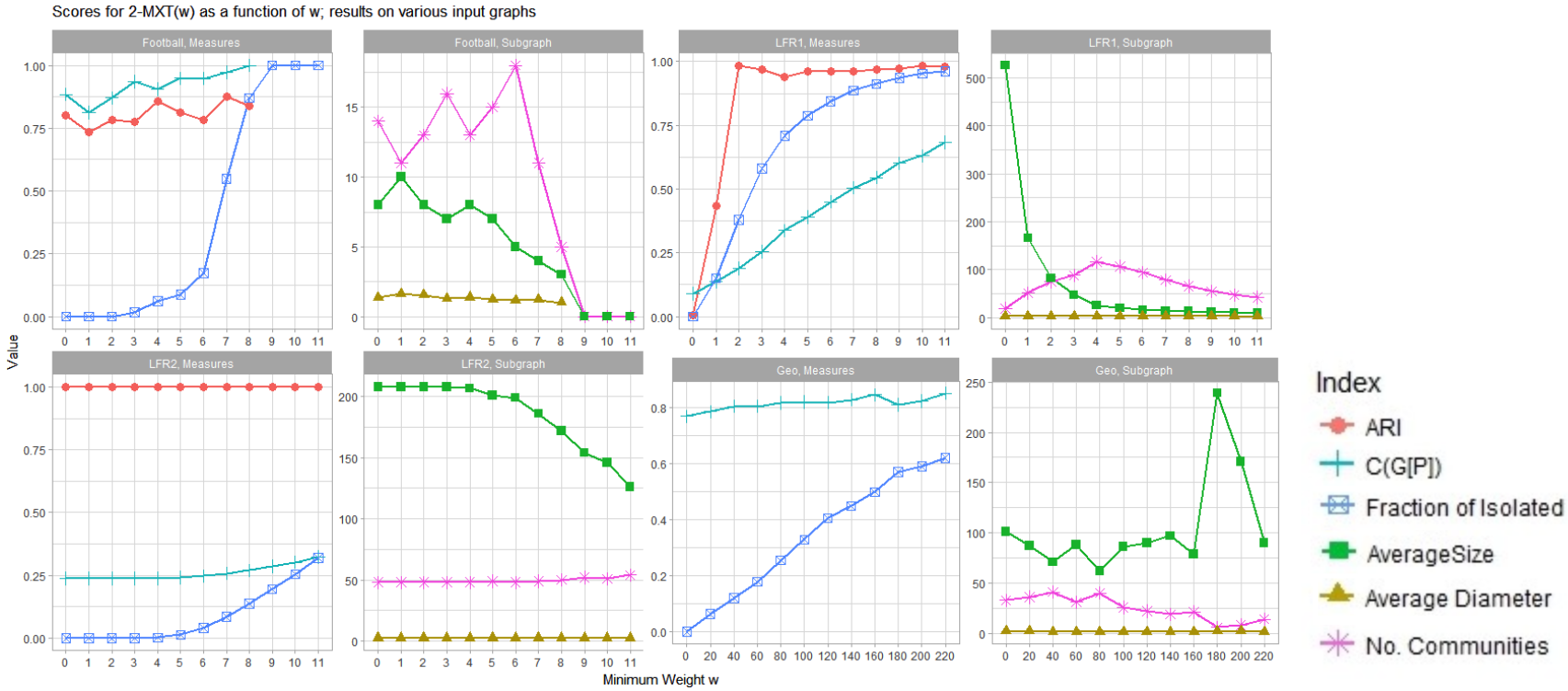
The right figure shows the: *number of communities, average community size* and *average diameter*. The average community size decreases with w, as more vertices are *isolated*. The number of community fluctuates around 15 and drops rapidly for $w \geq 7$, which corresponds to the sharp rise in the fraction of isolated vertices in figure (a). This is because, most edges in the graph have weight less than 7 i.e. see Figure 10.9 - (a) for its edge weight distribution. Finally, it appears that the resulting community-induced subgraphs have relatively low diameter.

It is seen that generally the number of communities varies only slowly with w. Moreover, as more vertices are isolated, the average size decreases. Notably, there is a spike in the average size of *geo graph*'s plot when $w = 180$, this is due to the decline in number of communities. There is also a sharp decrease of LFR1's average sizes when $w = 1$, which is the average edge weight. Finally, the community-induced subgraphs appear to have low average diameters in general.

Figure 10.12 shows the performance of $1,2,3,4,5$-MXT$(\bar{w})$ where $\bar{w}$ denotes the mean edge weight of the input graph. Overall, the results are similar across all test graphs. Particularly, as the parameter w is fixed, the fraction of isolated vertices remains a constant. That is, the number of isolated vertices remains fixed at approximately $0\%, 40\%, 25\%$ and 32% of $|V|$ for the football, LFR1, LFR2 and geo-tagged graphs, respectively. Furthermore, as k increases, the number of communities decreases and hence the average size also increases. Consequently, the average clustering coefficient also decreases.

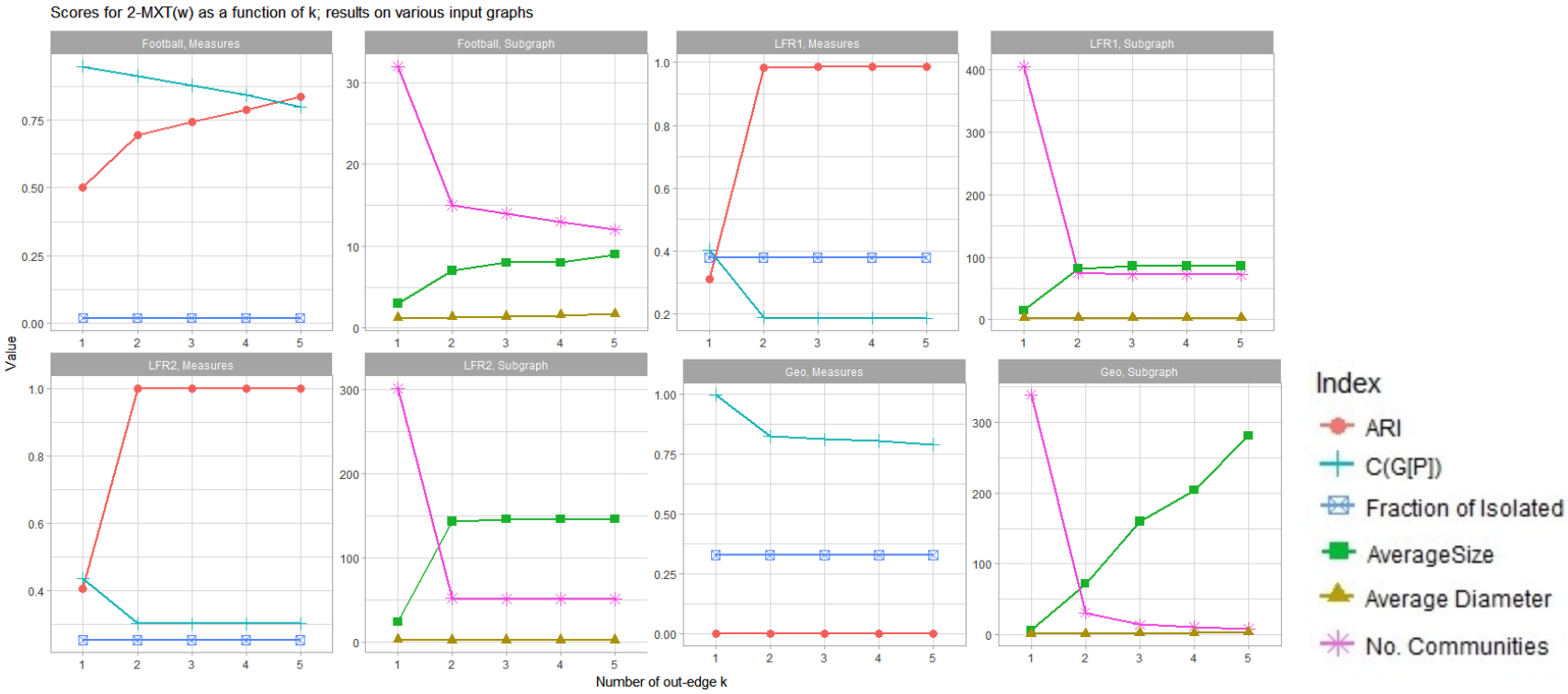
Notably, it appears that $k = 2$ is a threshold in the algorithm's performance. That is when $k \geq 2$, the algorithm produces significantly more accurate results, as indicated by the ARI scores (with the exception of the Geo graph, since there is no ground-truth community).

Compared to the rest, the *Geo* graph has a minor difference. That is, the average size grows linearly with k. This is due to the fact that the graph contains significantly more triangles. Hence, the fragments are still able to expand in size with increasing k.



(a) (b) Legend

Fig. 10.11 Performance of k-MXT(w) **with varying *w*** on four testing graphs: *football, LFR1, LFR2* and *geo*. In these experiments, *k is fixed at* 2 i.e. the algorithm used is 2-MXT(w). Furthermore, *the minimum weight w is increased* from 0 to above the mean edge weight. In these figures, we keep track of *six measures*, for which the graphs are plotted *in pairs* as a function of w. For example, the *two top-left figures* show the performance of 2-MXT(w) on the *football network*; see Figure 10.10 for a sample interpretation of this data. Note there are no *ground-truth communities* for Geo graph and hence the ARI curve is omitted.



(a) (b) Legend

Fig. 10.12 Performance of k-MXT$(\bar{w})$ **with varying *k* for fixed *w*** on four testing graphs: *football, LFR1, LFR2* and *Geo*. In these experiments, the minimum weight *w is fixed* at approximately the mean edge weight i.e. the algorithm used is k-MXT$(\overline{w})$. Here *the number of out-edges k is increased* from 1 to 5. In these figures, we keep track of *six measures*, plotted as a function of k. The plots are separated according to the testing graph and the type of the measure. For example, the *two bottom-right figures* show the performance of k-MXT$(\overline{w})$ on the *Geo* graph. Note that since Geo graph does not have *ground-truth* communities, we set $ARI = 0$ (red).

It appears that, $k = 2$ is a threshold by which the ARI on Football, LFR1, LFR2 indicates a sharp rise in accuracy. Looking at the *secondary plots*, the *pink* curves (number of communities) illustrate the tendency to break the partitions into many small fragments at $k = 1$. Transiting to $k = 2$, these fragments then *join together* to form the *correct* partition, hence $ARI = 1$.

The Geo graph seems to be a unique case. In the sense that *the average size increases* with k (hence the *number of communities* slightly decreases). This indicates the fragments *expand* with k. Whereas in LFR graphs, the fragments do not change for $2 \leq k \leq 5$. This is due to the fact that Geo graph contains a significantly higher number of triangles compares to the rest; hence, vertices have many unique *neighbours* to choose.

10.5.3.4 Remark.

It was discussed earlier that increasing k should *join up* communities together. This gives a sense of *hierarchy* for the results. Such that the fragments produced by small values of k are *nested* inside those produced by larger k. The same effect can be achieved by decreasing w.

The two parameters k and w have different intuitions. By increasing k, we *expand* the *current* communities by connecting communities together to create larger clusters e.g. see Figure 10.12. Whereas by increasing w, we *filter out* vertices with lesser triangle-density. Consequently, we condense the communities and its triangle-density increases e.g. see Figure 10.11.

We show an illustration of the hierarchical structure in Figure 10.13 below.

10.6 Conclusion

In this chapter, we studied the application of the triangle fragmentation algorithm k-MXT on real-world networks i.e. social and geographical networks. Overall, k-MXT seems to work well on real world networks if they are *sparse, but contain many triangles*. The parameter k is important. In many cases, the algorithm produce better results for $k = 2$. With $k = 1$ the results are often less *accurate* according to the ARI metric. However, this does not fully reflect the behaviour of the 1-MXT algorithm. On networks with ground-truth clusters, we observed that this variation tends to break each ground-truth community into small fragments which could be of size *two* i.e. a double edge, created when a pair of vertices point to each other.

The algorithm is sensitive to cross-partition edges. Thus by increasing k it is more likely to select these edges. To compensate for this, we introduced a parameterised variation k-MXT(w) and test it on a similar set of input graphs. The results indicate that

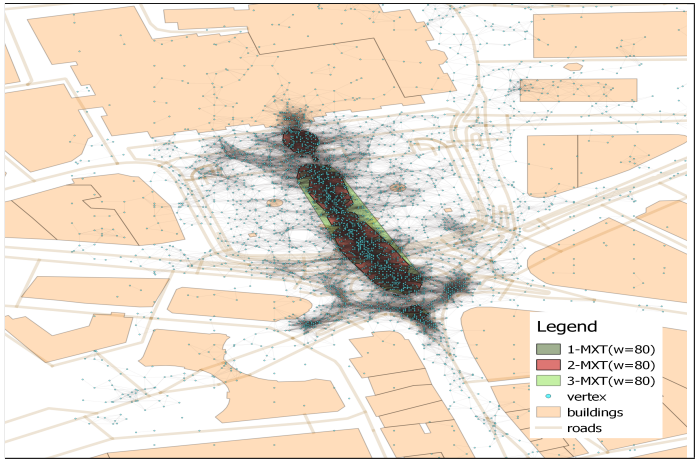
1. The community-induced subgraphs appear to have have high clustering coefficient and low diameter, which could perhaps be small-world subgraphs [59].

2. For networks in which ground-truth communities are present, the algorithm produces high *revised ARI* scores. This indicates the fragments lie inside its ground truth communities i.e. they are the subgraph of these communities.

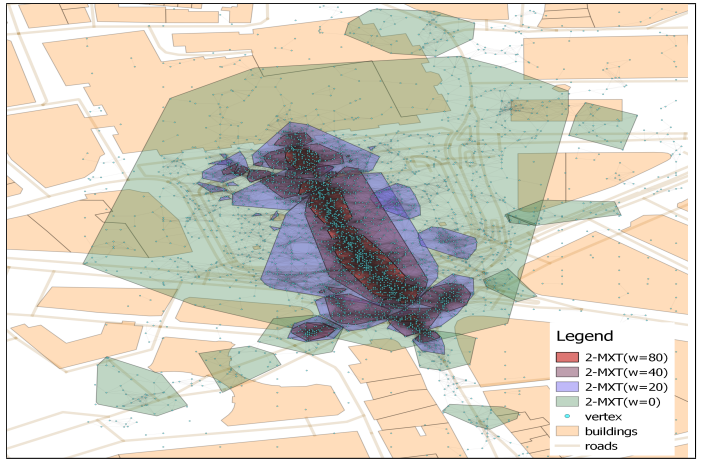
3. The parameter w is an edge filter. Thus, w should be chosen carefully to ensure a sufficient number of remaining vertices. Else the results might be less significant since there are many *isolated vertices*.

4. The parameter w and k give a naturally hierarchical structure to the clusters. Picturesquely, fragments can expand or shrink as the effect of varying these parameters.

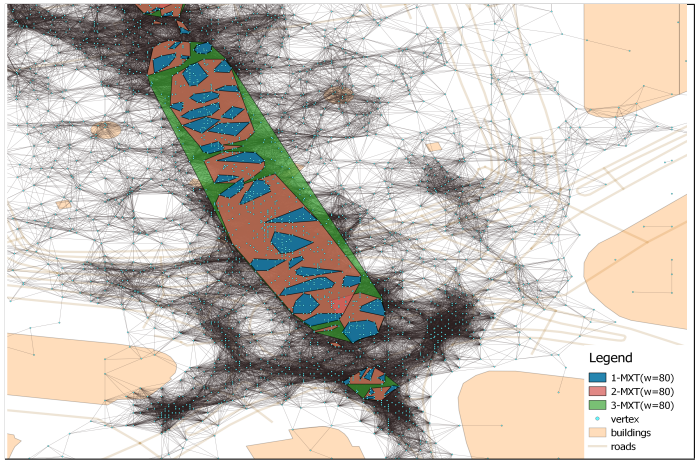
Experimentally, the k-MXT algorithm has good running time when the graph is sparse and the maximum degree in the graph is low. This, however, worsen when the maximum degree increases. See Section 12.6.2.2 for a detailed analysis of the algorithm's complexity.



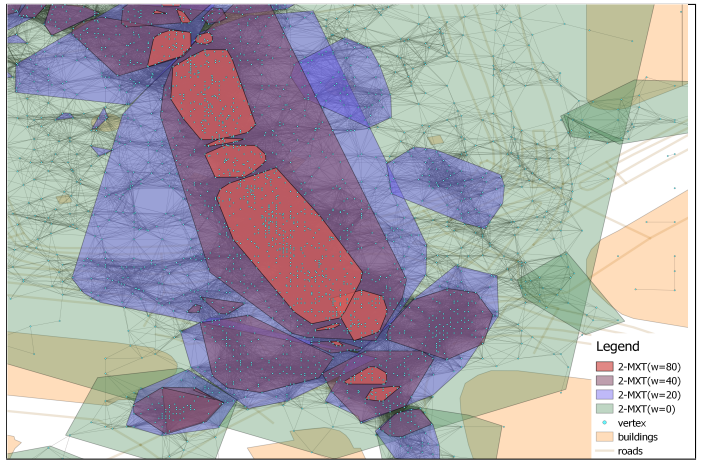
(a) k varies: full view



(b) w varies: full view



(c) k varies: zoomed



(d) w varies: zoomed

Fig. 10.13 *Clusters* produced by k-MXT(w). Figure (a) and (b) show a *full view* of the results produced by **varying k and w**, respectively. Figure (c) and (d) show a closer look at the structure of the clusters. Note that in Figure (b) and (d) *the edges are drawn more transparent* to make the clusters more visible. The example contains a fraction of the geo-tagged dataset. We replicate the experiments in the previous section. Particularly, the clusters produced here are the results of k-MXT(w) by fixing either w or k and increasing the other; the outputs are shown in Figure (a)(c) and (b)(d), respectively. It can be seen that the results display a *hierarchical* structure. The smaller *communities*, produced by e.g. a small value of k, are contained inside those produced by larger values of k.

Chapter 11

Experimental study of fragmentation on geometric graphs.

In preparation for the final chapter which will visit the original motivation of this study i.e. geo-tagged data, in this chapter we experiment with various fragmentation algorithms on geometric graph in order to gain some intuition about the algorithms in *geometrical settings*.

11.1 Geometric data

The original motivation of this study is to tackle the problem of finding popular places at which people photograph. Particularly, given a spatial dataset consists of geographical coordinates i.e. latitude and longitude, we wish to identify regions with high density i.e. contains large amount of coordinates. These regions are then, intuitively, the highly photographed places.

As each data point consists of only two features: latitude and longitude, the task can be considered as a problem of clustering points in *two-dimensional space*. Furthermore, in this setting, hypothetically there are underlying probability distributions of places where people taking photographs, with the means correspond to popular attractions, see Figure 11.1 for an example. Given only points generated by these hidden distributions, we are tasked with the objective of grouping together points which are likely to be generated by the same distribution.

Inspired by the above hypothetical scenario, we introduce some artificial datasets in which points are generated in 2D space according to some *planted* distributions. Using these points, we then construct our graphs by connecting every pair of vertices whose distance is at most some set constants. The result is then a *geometric graph*. These graphs are then input to our algorithms, to test if we can recover the *original clusters*.

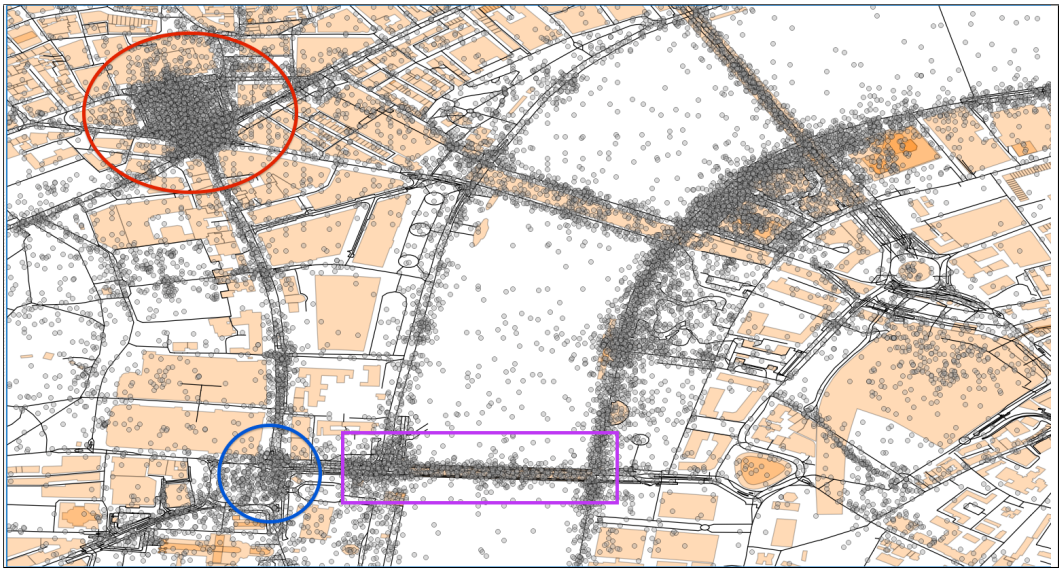


Fig. 11.1 The figure above shows the locations where photographs are taken in central London. Each dot pin-points a location where a photograph is taken, in geographical coordinates i.e. latitude and longitude. The picture consists of thousands of dots, plotted onto the map of London. It is seen that, there are some regions which are significantly *denser*, for instances: the *red circle*: Piccadilly circus, *blue circle*: Parliament Square Garden, the *purple rectangle*: Westminster bridge. These places are, certainly, popular *landmarks* where visitors like to take photographs. One can imagine that, around each landmark, *dots* are distributed according to some distribution and constraints e.g. range of visibility, physical constraints, etc.

11.2 Artificial dataset

11.2.1 The 'mouse' dataset

The 'mouse' dataset [33] is a synthetic dataset is often used in testing various clustering algorithms. The name comes from the resulting shape of the dataset, which resembles the famous cartoon character (see Figure 11.2). It is generated as follows.

Consider three circles C_1, C_2, C_3, centred at c_1, c_2, c_3 with radius r_1, r_2, r_3, respectively. Let C_1 be the largest circle while C_2 and C_3 are of equal sizes, specifically $r_1 = 2.r_2$ and $r_2 = r_3$. The centres are then positioned so that C_1, C_2 and C_1, C_3 are tangent while C_2 and C_3 are separated i.e. $||c_1, c_2|| = r_1 + r_2$ and so on.

These circles are now our *boundaries* inside which we will generate points according to some distribution with its mean being the circles' centres. For each circle C_1, C_2, C_3 we put n_1, n_2, n_3 points inside them, respectively. As C_1 is double the size of the others, we put $n_1 = 2.n_2$ and $n_2 = n_3$. One can make the test more difficult by having the circles intersect. However, the task is already difficult with the tangent circles.

Next, we choose the distribution. The most popular option is the Gaussian distribution. For each point p in circle C_i, we sample each coordinate x_p and y_p independently, from the Gaussian

(a) *Gaussian* mouse (b) *Uniform* mouse

Fig. 11.2 The 'mouse' dataset, generated by two different distributions. On the left: Gaussian or normal distribution. On the right: uniform distribution. The total number of points is $4,000$ with $n_1 = 2,000$ and $n_2 = n_3 = 1,000$. The vertices/points are coloured-coded according to its distribution.

distribution i.e. $\mathcal{N}(\mu, \sigma^2)$ with mean μ and variance σ^2. That is,

$$x_p \sim \mathcal{N}(x_{c_i}, r_i/2) \qquad\qquad y_p \sim \mathcal{N}(y_{c_i}, r_i/2),$$

where x_{c_i}, y_{c_i} is the coordinates of the centre c_i and r_i is its radius.

We also consider the uniform distribution. Consider two uniformly distributed variables $r \in [0,1]$ and $\theta \in [0, 2\pi)$. For each point p inside circle C_i its coordinates are then given by [60]

$$x_p = \sqrt{r}\cos(\theta) \times r_i + x_{c_i} \qquad\qquad y_p = \sqrt{r}\cos(\theta) \times r_i + y_{c_i}.$$

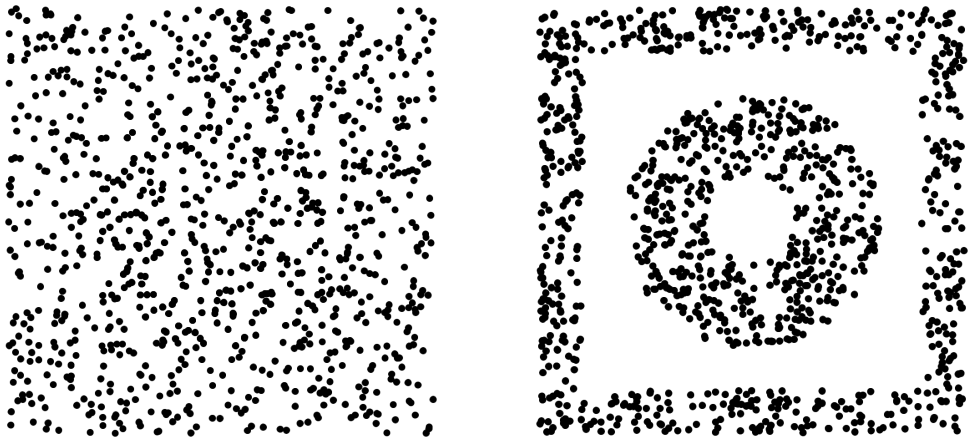
Figure 11.2 shows the examples of the 'mouse' datasets.

11.2.2 Points in a unit square

In the previous 'mouse' dataset, there exists a sense of *natural clusters* i.e. the distribution whence each point is sampled from. We then question: what if the input data does not have any natural clusters? What would the algorithms do in such cases? And hence, to investigate this we apply the algorithms on a *neutral* dataset, in which points are generated randomly in a unit square in two-dimensional space. In other words, we generate n points, where the x, y coordinates of each point p is sampled uniformly at random in $[0,1]$.

11.2.2.1 Shapes in a unit square.

Given points generated u.a.r in a unit square, we can *trim* some points to create some specific shapes, see for example Figure 11.3. It is also interesting to see if the algorithms are able to reproduce these shapes.



(a) Points in unit square (b) Shapes in unit square

Fig. 11.3 The left figure shows $1,000$ points generated randomly in a unit square. Removing a set of points give us the shapes in the right figure.

11.2.3 Graph construction using spatial points

Given the spatial datasets, we construct the input graphs by connecting every pair of vertices (v,u) such that

$$dist(v,u) \leq d,$$

where $dist(v,u)$ is a *distance* function and d is a specified constant, the *distance parameter*. Typically we use the Euclidean distance i.e. $dist(v,u) = \sqrt{(x_v - x_u)^2 + (y_v - y_u)^2}$. In other words, for each vertex/point v we connect v with every vertex u which lies within a circle C_v with radius d, centred at v.

11.2.4 Some theoretical foundation of random geometric graph

When input points are generated in a unit square i.e. as in Section 11.2.2 and see for example Figure 11.4, the resulting graph is often known as a *random geometric graph* (RGG) i.e. $G(n,d)$ where n is the number of points and d is the radius of each circle. The RGG model is widely studied to model spatial networks [15] which mainly focuses on connectivity [12] [48] and continuum percolation [48]. Below we review some main points regarding the connectivity of the RGG model on *two-dimensional space*.

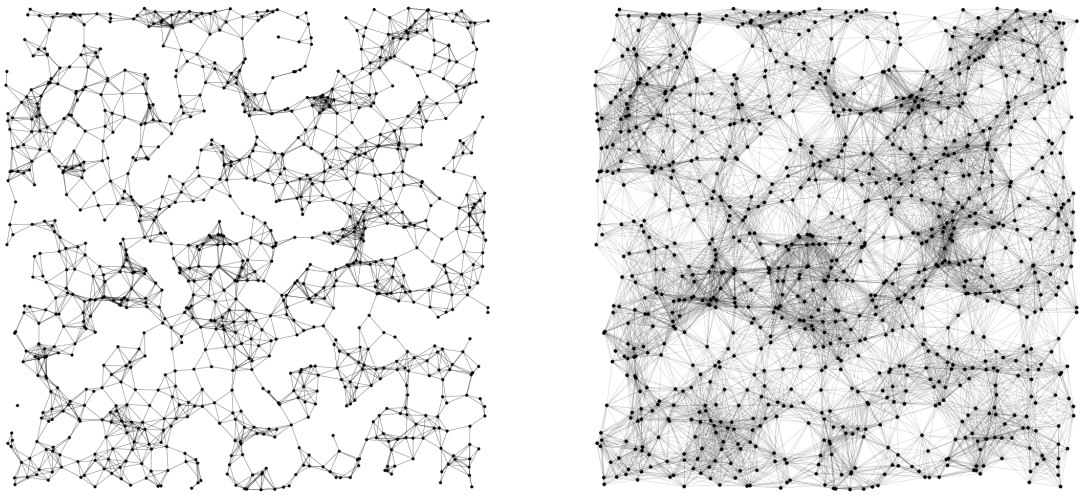
For convenience let us replace the radius of each circle by r instead of d as in previous sections. By definition, each vertex v is connected with every vertex u within a circle with radius r, centred at v. Let us ignore the case where vertex v lies close to the boundaries that its circle is blocked. The probability that a vertex u *falls in* the circle of v is then the circle area i.e.

$$Pr(u \text{ connect } v) = \pi r^2, \tag{11.1}$$

and the complimentary event is

$$Pr(u \text{ does not connect to } v) = (1 - \pi r^2). \tag{11.2}$$

Since it is required that $\pi r^2 \leq 1$, it is assumed that r is small enough. In general, we can choose $r = r(n)$ to control this radius parameter.



(a) $r = 0.05$ (b) $r = 0.1$

Fig. 11.4 A two-dimensional random geometric graph in a unit square with $n = 1,000$. On the left: $r = 0.05$. On the right: $r = 0.1$. The graph on the left is not connected as there is one visible isolated vertices i.e. bottom-left corner. The graph on the right is connected and relatively dense.

As there are n vertices are distributed uniformly and independently in the unit square, the expected degree of a vertex v is proportional to the area of the bounding circle, that is

$$E[d(v)] = n\pi r^2. \tag{11.3}$$

From (11.2), consider the event that vertex v is isolated i.e. $d(v) = 0$ occurs with probability

$$Pr\{d(v) = 0\} = (1 - \pi r^2)^{n-1} \leq e^{-\pi r^2 (n-1)},$$

where the inequality comes from: $(1-x) \leq e^{-x}$ for $0 < x < 1$. If we let $\pi r^2 n = c \log n$, then

$$Pr\{d(v) = 0\} \leq e^{-c \log n(1-1/n)} = n^{-c(1-1/n)} \sim n^{-c}. \tag{11.4}$$

Let X_i be an indicator random variable that assumes value 1 if vertex v is isolated and 0 otherwise; and let X be its sum i.e. the number of isolated vertices in G. By linearity of expectation, the expected number of isolated vertices is then

$$E[X] = \sum_{i \in V} X_i = n(1-\pi r^2)^{n-1} \leq n e^{-(\pi r^2 (n-1))} \sim n^{1-c}. \tag{11.5}$$

For a graph G to be connected, having no isolated vertices is a *necessary condition*. However, this condition is not always sufficient for G to be connected. More particularly, it can be seen that

$$Pr(\text{G is connected}) \leq Pr(\text{G has no isolated vertex}).$$

It turns out that the latter is *asymptotically sufficient* in order for a RGG to be connected. Specifically, Penrose [48] shows that as $n \to \infty$, the two distributions have the same limiting behaviour. In other words

$$\lim_{n \to \infty} Pr(\text{G is connected}) = \lim_{n \to \infty} Pr(\text{G has no isolated vertex}).$$

This leads to the following theorem, due to Penrose [48], Gupta and Kumar [25],

Theorem 9 (Connectivity of RGG)**.** *Let G be a RGG in a unit square. Let X_i be an indicator variable such that $X_i = 1$ iff vertex v_i is isolated. Let X be the number of isolated vertices in G, $X = \sum_i X_i$. Let $\pi r^2 n = c \log n$, then*

$$E[X] = \sum_{i \in V} X_i = n(1-\pi r^2)^{n-1} \leq n e^{-(\pi r^2 (n-1))} \sim n^{1-c}.$$

It follows that

1. *If $c > 1$, then, $E[X] \to 0$ and G is connected w.h.p;*
2. *If $c < 1$, then, $E[X] \to \infty$ and G is disconnected w.h.p.*

Furthermore, if $r = \omega/\sqrt{n}$ then G has a giant connected component.

11.2.4.1 Triangles in RGG.

Take a vertex v, in order to form a triangle e.g. (v,u,w) vertex u and w must *fall* within the circle of v. Since u and w are sampled independently, from (11.1) we get

$$Pr\{(v,u,w)\} = \pi^2 r^4.$$

Let the expected number of triangles that contain v be $\triangle_v$. As there are $\binom{n}{2}$ ways to select u, w, we get

$$E[\triangle_v] = \binom{n}{2}\pi^2 r^4 = \frac{n(n-1)}{2}\pi^2 r^4 \approx n^2\pi^2 r^4.$$

If $r = \omega/\sqrt{n}$ for some fixed ω then

$$E[\triangle_v] \approx n^2\pi^2 r^4 = \pi^2\omega^2. \tag{11.6}$$

On the other hand, consider the random graph $G(n,p)$ model. Denote by $\triangle'_v$ the expected number of triangles contain the vertex v (we have studied this in Section 9.1.1). Consider $p = \omega/\sqrt{n}$, we have

$$E[\triangle'_v] = \binom{n}{2}p^3 \approx n^2 p^3 = \omega^2 n^{-1/2}, \tag{11.7}$$

Comparing the expected degree for each vertex v and hence the expected number of edges of the of the two models, we have for RGG

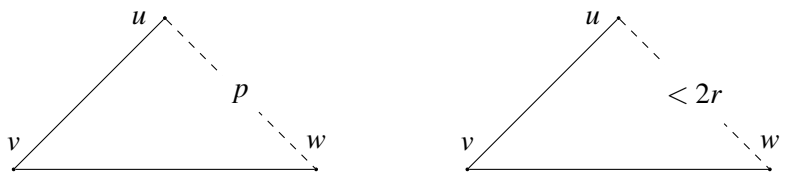
$$E[d(v)] = n\pi r^2 = \pi\omega^2 \quad \text{thus} \quad E[\#\text{ edges in RGG}] = \sum_v E[d(v)] = n\pi\omega^2,$$

compares with $G(n,p)$

$$E[d'(v)] = np = \omega\sqrt{n} \quad \text{thus} \quad E[\#\text{ edges in } G(n,p)] = \sum_v E[d'(v)] = \omega n^{3/2}.$$

Thus, it is seen that the $G(n,p)$ model has lower *triangle density* i.e. (11.6) compares to (11.7), despite having a much higher *edge density*.

This difference makes an intuitive sense because in $G(n,p)$ the event v is connected to u and w does not affect the probability that u and w are connected. On the other hand, in geometric graphs, given the event that v is connected to both u and w, it is likely that u is also located near to w. Further, if the vertices are embedded in Euclidean space, then the distance between u and w is at most $2r$, by the triangle inequality. Having many *triangles* is therefore considered a fundamental property of geometric graphs, which differs it from random graphs [53]. There is, in fact, an increasing interest in extracting knowledge from input graphs by uncovering its underlying geometric setting (if exists) see e.g. [53].



11.3 Experiments

For all datasets, we generate $n = 4,000$. Furthermore, for the 'mouse' dataset, we set the radius of the the largest circle at 200, and 100 for the two smaller circles 100.

Beside the k-MXT algorithm, we also consider different versions of the fragmentation algorithm. We begin by introducing these versions.

11.3.1 Variations of fragmentation algorithm

Recall that the core procedure of the fragmentation algorithm consists of the following steps:

1. select an *unmarked* vertex v;
2. select k neighbours of v, subject to some functions;
3. direct k edges from v to the selected neighbours;
4. mark the vertices, according to set rules;
5. repeat.

For step 2, in previous sections, we studied some versions in which typically $k = 1$ and the neighbour is selected at random. For step 4, we usually consider two variations where the main difference is which vertex to *mark*. Particularly, for a pair of vertices (v,u) where v is the *selected* vertex and u is the *neighbour*, if it is stated that

- *v absorbs u*: then u is marked;
- *v points to u*: then v is marked.

Steps $2,3$ and 4 together make up the *mapping* process. The process can easily be changed to create different versions of the fragmentation algorithm. Thus, the following are the variations that we consider for this experiment i.e. see Table 11.1.

11.3.2 Evaluation

To evaluate the algorithms performance on the *mouse dataset*, we consider the distributions or the original *circles* $C = \{C_1, C_2, C_3\}$ the *ground-truth clusters*. For the *shapes in unit square* dataset, we consider the two shape in Figure 11.3 (b) as the ground truth. Recall that the output of the algorithms is a set of connected components $K = \{K_1, \ldots, K_i\}$ where $i \geq 1$. To evaluate the results, the common strategy of comparing the *ground-truth label* and the *result-label* is not ideal. This is because the definition of *labels* here is not well-defined i.e. the indices of the ground-truth clusters and the resulting components can be change arbitrarily.

Therefore, we evaluate the similarities of the set C and K using the Adjusted Rand Index (ARI). As introduced in Section 7.3.1, the ARI measures the similarities of two given sets by counting the

Algorithm	Abbreviation	Mapping rules
Absorb Nearest Neighbours	abNN	Pick a vertex v u.a.r; v *absorbs* its k *nearest* neighbours.
Absorb Random Neighbours	arn	Pick a vertex v u.a.r; v *absorbs* k *random* neighbours.
k Nearest Neighbour	kNN	For each vertex $v \in V$; v points to its k *nearest* neighbours.
Permutation Subgraph	permSub	For each vertex $v \in V$; v points to k *random* neighbours.
k Max Triangulation	k-MXT	For each vertex $v \in V$; v selects k densest triangles, see Section 8.1 for details.

Table 11.1 Fragmentation algorithms in comparison

number of pairs of vertices which are classified in the same and different clusters in the two given sets. Thus the ARI is applicable without any labels. The index ranges in $[-1;1]$ where scores closer to 1 indicate the two sets are highly similar e.g. a score of 1 indicates an exact match. Thus, a high ARI score indicates the output closely resemble the original clusters.

Evaluation on the RGG might be more problematic since there is no notion of *natural clusters*, which is the motivation of the dataset. Thus, we resort to visual inspection.

11.4 Results

We present the experimental results in this section. For each algorithm, we consider different values of k. Particularly, we set $k = [1,5]$ we execute each algorithm 10 times and take the average score. Finally, we plot the scores as a function of the distance parameter d.

The figures below presents the ARI scores for the two 'mouse' datasets and also the *shapes in unit square*. It is expected that the 'uniform' data would be more challenging compared to the 'Gaussian' dataset. This is arguably apparent from the overall scores i.e. visually, the average scores are lower in (b) than in (a). Overall, based on the ARI scores, the algorithms can be divided into two groups: *kNN*, k-MXT and permSub with better performance; *abNN* and *arn* produce lower performance.

It makes intuitive sense that the distance d is an important parameter. The last three algorithms i.e. kNN, k-MXT and permSub have its peak performance for the *right* value of d. On the Gaussian 'mouse', the peak performances are considerably good for k-MXT and permSub. However, on the uniform 'mouse', all best scores are below 0.5. Finally, for the *shapes* dataset, the algorithms can produce the *correct* clustering for some given parameters.

(a) The figures above show result on the **Gaussian mouse** dataset. The curves show the ARI scores as a function of the distance d. Each curve corresponds to a unique value of $k = 1, \ldots, 5$. The algorithms are, from left to right, *abNN, arn, kNN, kMXT* and *permSub*; as indicated by its label.

Recall that a *high score of ARI* indicates the testing algorithm produces a clustering similar to the input. Hence the algorithms can be ranked (from best to worst) based on its performance as follows: k-MXT, permSub, kNN, arn and abNN. With correct value of d, the k-MXT is able to produce accurate clusters. The accuracy drops as d increases as more *inter-partition* edges are selected. Thus, higher values of k are even more sensitive to the distance parameter.

Surprisingly, with $k = 1$ the permSub produces relatively good result. The reason is: as d increases, the size of the components (which are cycles) produced by permSub also increases. With high value of d, the input graph is much denser, thus permSub is able to find *longer cycles* which span within the correct *clusters*. Furthermore, given that the input clusters are quite separated e.g. see Figure 11.2-(a), there are not many *wrong* vertices included in such paths. See examples below.

(b) permSub: $k = 1$, $d = 10$ (c) permSub: $k = 1$, $d = 95$ (d) 2-MXT: $d = 35$

Fig. 11.5 Results on the **Gaussian mouse** dataset.

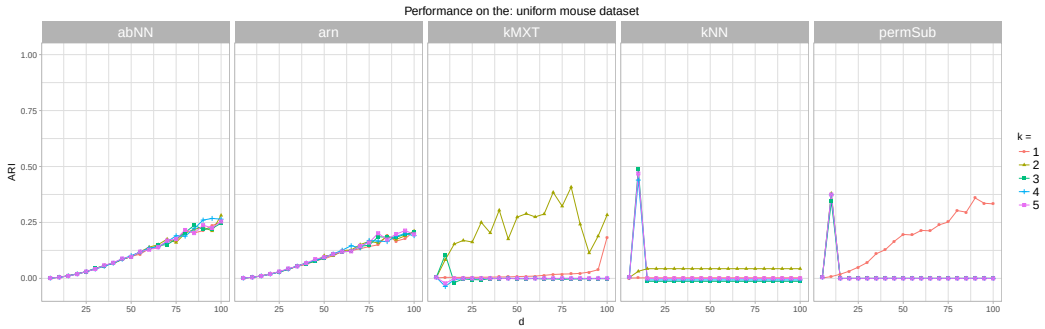


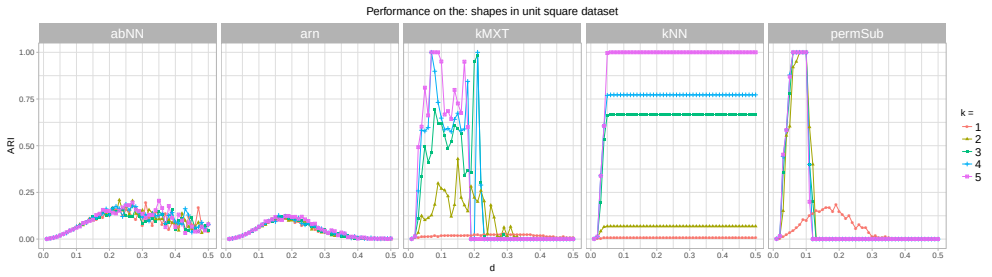
Fig. 11.6 Results on the **Uniform mouse** dataset. The algorithms can be roughly ranked based on its performance from best-worst: kNN, permSub, k-MXT, arn and abNN. This experiment is more difficult to achieved higher score, compares to the Gaussian mouse.

Fig. 11.7 The results on the **shapes in unit square** dataset. The algorithms can be ranked based on its performance as follows: kNN, k-MXT, permSub, arn, abNN.

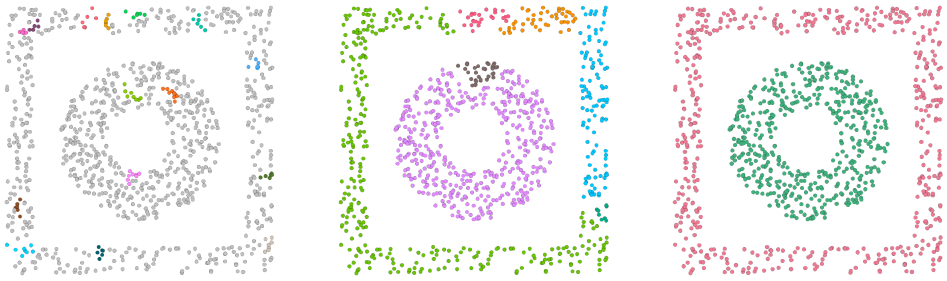
In this test, the two clusters are very well separated. Hence with the *correct* values of d i.e. d is less than the minimum distance between any pair of vertices between the clusters. The algorithms will not choose any incorrect edges. Thus, to achieve high ARI scores, the task is to *fully-connect* vertices within each cluster, which can be achieved by increasing k. This is evident from the performance of permSub. That is, for approximately $0.05 \leq d \leq 0.15$, permSub achieve high scores with $k \geq 2$.

For kNN, we see that the deterministic behaviour of the algorithm, see Figure 11.8 for an illustration of the graphs generated by kNN with d fixed and k increases.

The k-MXT achieves good results for $d \leq 0.2$ with $k \geq 2$. For $d \geq 2$, the graph is dense, thus the algorithm returns 1 component (with the exception of $k = 1$).



The *kNN* algorithm has a deterministic behaviour. This is because, for each vertex its k nearest neighbours is determined. The algorithm appears to be very successful in the *shapes* experiments. The reason is that if the original clusters are well separated, then we only have to increase the *distance parameters* to increase the connectivity of the connected components, which effectively *expand* the resulting clusters, see for example Figure 11.8. This also explains why the algorithm permSub works well in this case. This strategy, however, fails in cases where points in different clusters are in close proximity i.e. clusters are not well-separated like in the cases of the 'mouse' datasets.

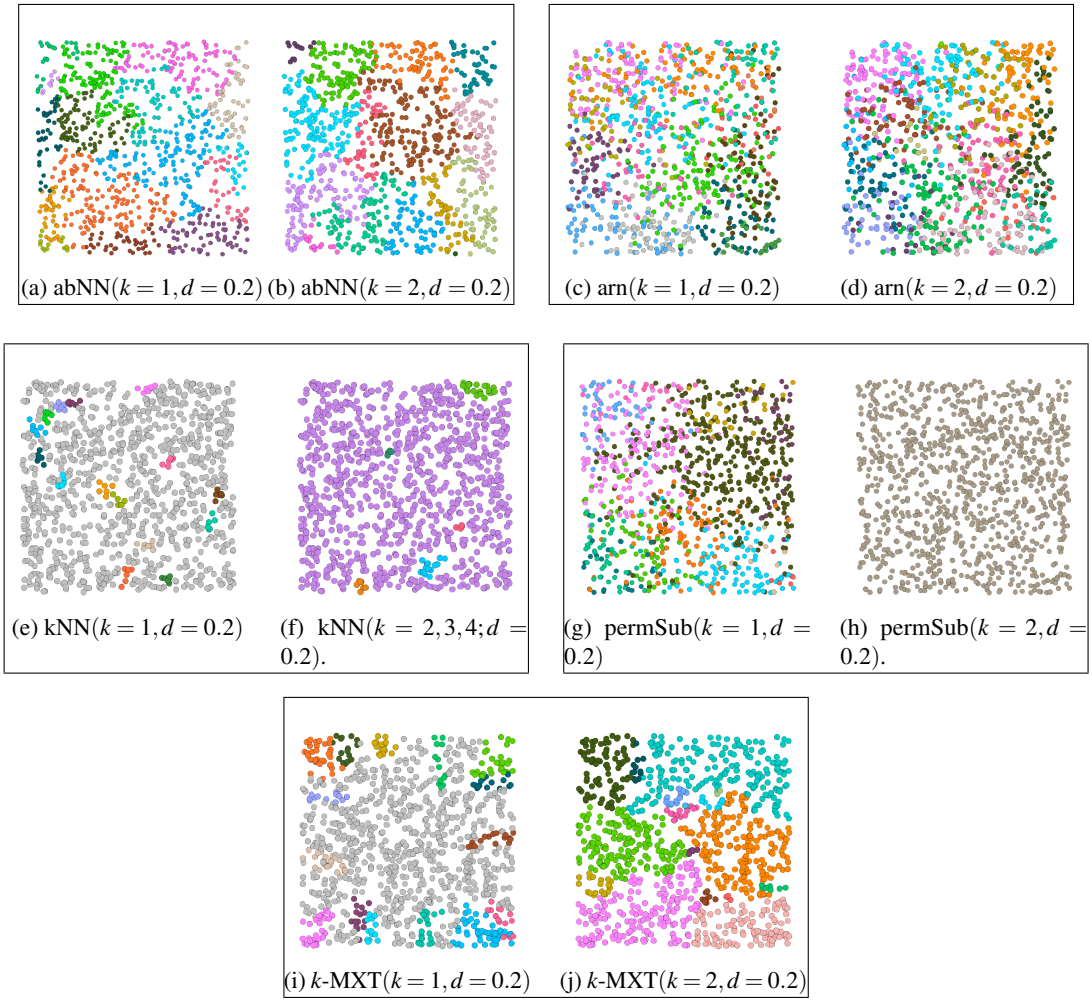


(a) kNN: $k = 1, d = 0.5$ (b) kNN: $k = 3, d = 0.5$ (c) kNN: $k = 5, d = 0.5$

Fig. 11.8 kNN with $d = 0.5$ and $k = 1, 3, 5$. The pictures show the expansion of components by fixing d and increasing k. Note that in Figure (a), there are many clusters. Hence, only some of the largest clusters are coloured while the rest are drawn in grey.

For experiments in RGG, the intention is to investigate the behaviour of the algorithms given densely connected geometric graph. This is motivated from the observation that in *real geographical*

graph i.e. obtained from photograph coordinates are dense. Furthermore, in some cases, attractions are located in close proximity e.g. Parliament Square Garden and House of Parliament (see Figure 11.1). Hence if we observe the data in these regions at the right level, the *dots* picture would look quite similar to a RGG. In which case, we want to test if the algorithm is able to produce *separated regions*.



(a) abNN($k=1,d=0.2$) (b) abNN($k=2,d=0.2$) (c) arn($k=1,d=0.2$) (d) arn($k=2,d=0.2$)

(e) kNN($k=1,d=0.2$) (f) kNN($k=2,3,4;d=0.2$). (g) permSub($k=1,d=0.2$) (h) permSub($k=2,d=0.2$).

(i) k-MXT($k=1,d=0.2$) (j) k-MXT($k=2,d=0.2$)

Fig. 11.9 Visualisation of the outputs on the *random geometric graph*. For these figures, the distance is set at $d=0.2$ and $k=1,2$. In each figure, the vertices are colour-coded according to its resulting cluster. Note that *there is a single cluster in figure (h)*, which is coloured in dark-grey. Furthermore, only the 15 largest clusters are coloured; the remaining clusters are coloured in grey. Each *frame* groups together two outputs of the same algorithm with different parameters.

It is seen that the *abNN* and *k-MXT* produce relatively well-separated and consistent fragment. However, there are some tiny fragments contained within larger clusters. The *kNN* is deterministic as noted, the *same result* in Figure (f) is produced different values of $k=2,3,4$. The *arn* and *permSub* cluster assignments seem rather random for $k=1$.

11.5 Conclusion

In this chapter, we studied experimentally the application of some graph fragmentation algorithms on geometric graphs. The graphs that we considered are constructed by connecting points generated randomly in 2D space. The construction of the graphs is motivated from a *real-world* dataset consists of geographical coordinates of the locations where photographs are taken.

For each input graph, we rank the algorithms based on each's performance, from best-worst, as follows

- *Gaussian mouse:* k-MXT, permSub, kNN, arn and abNN.
- *Uniform mouse:* kNN, permSub, k-MXT, arn and abNN.
- *Shapes in unit square:* kNN, permSub, k-MXT, arn and abNN.
- *Random geometric graph in unit square (based on the separation of clusters):* k-MXT, abNN, kNN, arn and permSub.

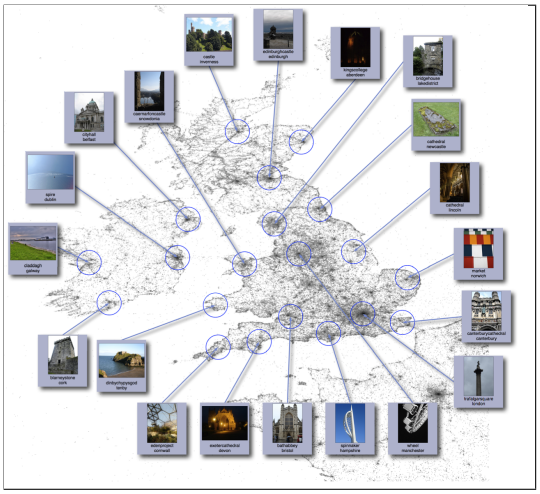
Overall, k-MXT arguably has the best performance out of all variations. Follows by permSub and kNN, the former works surprisingly well in some settings and the latter's behaviour is rather deterministic. The abNN and arn do not perform well objectively; however there could be some interesting properties related to its seemingly random cluster assignments as seen from Figure 11.9.

The experiments conducted in this chapter provide an intuition of the problem that we are going to present in the next chapter, in which we tackle the original motivation of the thesis.

Chapter 12

Application on geo-tagged data

Our original interest in the topic of this chapter came from studying the paper "Mapping the world photos" by Crandall et al [10]. In this paper, the major cities of the world are identified and tagged using *mean shift clustering*. The results for the UK are shown below [10]



Based on this, we collected geo-tagged photos from Flickr and made an initial investigation by plotting the geo-tag coordinates on the Earth's surface see Figure 12.1.

12.1 The dots

What we found really intriguing looking at these simple pictures of dots (see Figure 12.2) is how quickly our minds identify the *dark, dense* regions of dots; recognise the patterns and come to a reasonable conclusion that the subject of the picture is, probably, something that might be quite familiar to us.

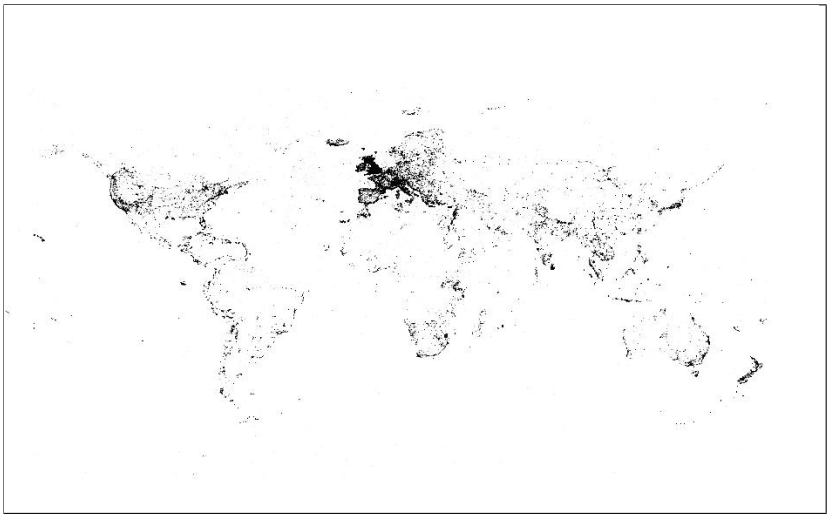


Fig. 12.1 World map created by geo-tagged photos.

Our guess is that, perhaps, our minds play a game of *grouping the dots*, and finish extraordinarily fast. Clearly, relevant prerequisite knowledge is required to recognise the subject. Otherwise, even though the *dots* might be *connected*, the answer is uncertain e.g. see Figure 12.3.

The *dots* from Figure 12.2 and 12.3 are GPS coordinates extracted from photographs, recorded as photographers and travellers snapping pictures from every corner of *certain cities*. Therefore, these pictures, created by thousands of *dots*, reflect interesting patterns of human collective behaviour particularly when travelling i.e. the dense regions are probably the most photographed attractions or popular observation points whence landmarks can be observed more attractively.

In this chapter, we are interested in studying an application of the triangle-fragmentation algorithm. More specifically, we apply the k-MXT on this spatial dataset with the motivation to identify sufficiently dense regions of dots. Such motivation is inspired by the observation that if we construct

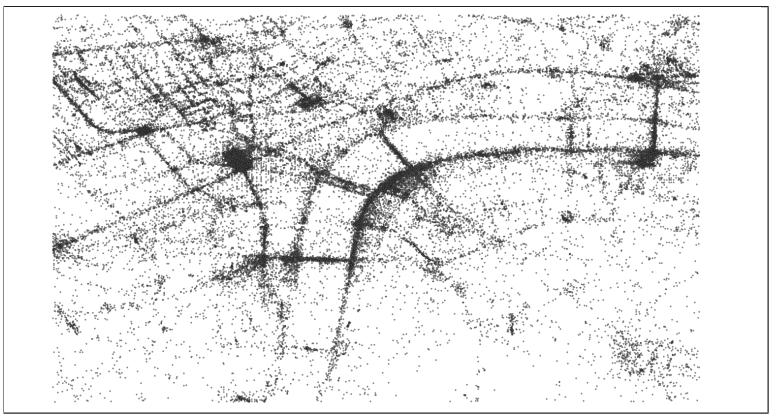


Fig. 12.2 A picture made of purely *dots*, can you recognise the subject of this figure?

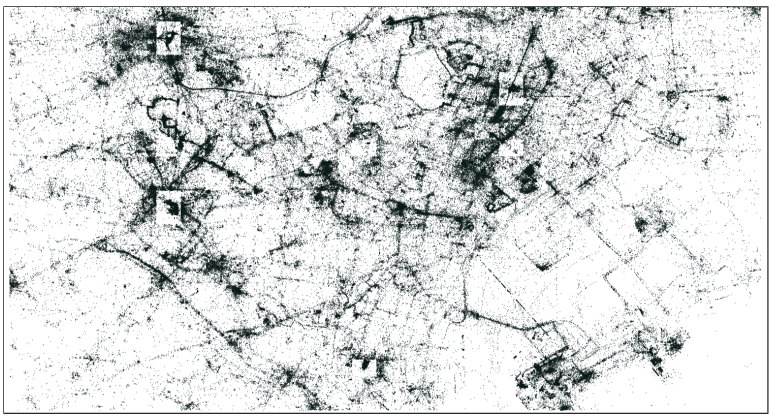


Fig. 12.3 How about this one? Answer: see Figure 12.22.

geometric disc-graphs using these data points as input, for a *correct* distance value, the subgraphs formed in these dense regions would contain many many triangles; thus applicable for the k-MXT.

In some sense, we are trying to replicate what, hypothetically, our minds are doing: *grouping the dots*. Of course there are tremendous difficulties: the immense amount of noise, approximate location generated from GPS etc. The problem is challenging but nonetheless attractive.

12.2 Data Collection

There are billions of geo-tagged photos, so an enormous corpus of *dots* that can be used to generate datasets. For this study, we select Flickr as the main photo-sharing platform as it is easily accessible and possesses a large database of geo-tagged photos.

For data collection, since Flickr imposes a limit on *the number of unique results* for each query with the same parameters, we used two strategies. The first is done by identifying a *bounding rectangle* covering the desired region. The bounding rectangle is further divided into x^2 unit rectangle for some desired number x. Finally, for each *unit bounding box*, we issue a query to collect photographs taken within.

The second strategy is done by *crawling* on the Flickr's *users' network*. We wish to collect as many *dots* in a specific city, for example, London. Thus, we start from users among a public group, named *'London by Londoners'*. From there, we perform a probabilistic Breadth First Search as follows. For each user, enqueue if the user's current location is set at London; otherwise the user is still enqueued with probability 0.01; while the rest are discarded.

Both strategies work well with certain pros and cons. The former is faster, however generates squares which have *abnormal density* (see Figure 12.3, top left). This occurs when there are a large number of data points (> 4000) within the *unit rectangle*. The latter is relatively slower, but it also gathers additional dimensions of data i.e. an underlying network of Flickr's users.

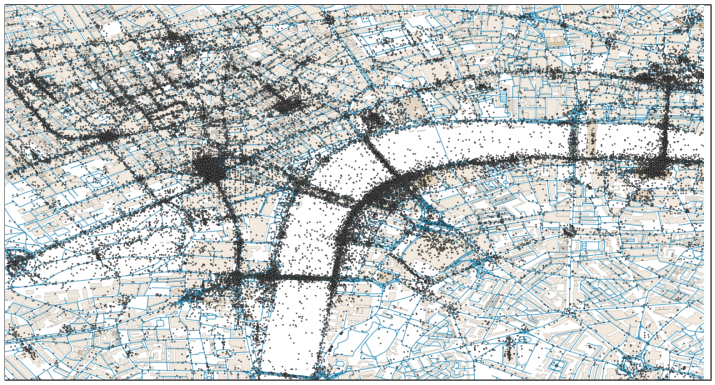


Fig. 12.4 Revealing the subject of Figure 12.2. The figure shows, approximately, 45,000 geo-tagged photos taken in various places in central London. The *dots* cover a region of roughly 15 km^2 square kilometres. The bounding rectangle has a height of 3*km* and a width of 5*km*.

12.2.1 Metadata

For each photograph publicly available for download, Flickr's API returns the following metadata

```
{   ...
    "title": "Portrait",
    "tags": "portrait england man london hat season ...
    "latitude": 51.10123,
    "longitude": -0.12974,
    ...
},
```

where

- *geo-spatial* i.e. latitude and longitude and can be empty;
- *tags* is the photo's set of textual tags, entered by the user;

in which, we are mostly interested in the *geo-spatial* metadata to generate our *dots* pictures.

12.3 Algorithms

In this section, we introduce the algorithms which are used in this experiment. The algorithms are: *mean shift*, DBSCAN and *k*-MXT. These algorithms share an important feature that is only a *distance parameter* is required. Furthermore, since they do not require in advance a number of clusters or the underlying distribution of data, the algorithms are most suitable for this problem.

12.3.1 Mean shift

Mean shift is a non-parametric technique for locating the maxima of a density function [7]. It is used as the method of choice for clustering in the paper "Mapping the world photos". Mean shift follows an iterative procedure to locate the modes of an underlying probability distribution given a set of sample points. As mentioned, the advantage of the mean-shift algorithm is that it requires only a *distance parameter h* i.e. the radius of the circle surrounding each data point.

The mean-shift procedure is computed as follows. Let $P = \{p_1, p_2, \dots p_n\}$ be the set of input points. For each point p_i, let C_i be a circle with radius h with p_i being its centre. Given C_i, we locate all points p_j lie within C_i, equivalently p_j are points from which the distance between p_i and p_j is less than h i.e. $dist(p_j, p_i) \leq h$, using any distance function. Then, using every point $p_j \in C_i$, we calculate the *weighted mean*

$$m(p_i) = \frac{\sum_{p_j \in C_i} \{p_j . f(p_j, p_i)\}}{\sum_{p_j \in C_i} f(p_j, p_i)}, \tag{12.1}$$

where $f(p_j, p_i)$ is a kernel function that determines the weight of nearby points for re-estimation of the mean; hence $p_j.f(p_j, p_i)$ multiplies the coordinate vector p_j with the scalar f. We use the Gaussian kernel

$$f(p_j, p_i) = e^{-\{dist(p_j, p_i)\}^2 / 2h^2}. \tag{12.2}$$

The vector $m(p_i) - p_i$ is called the *mean shift vector*, and the initial data-point is then *shifted* to $m(p_i)$ i.e. $p_i \leftarrow m(p_i)$ and then continue recursively for p_i. The recursion stops if the mean converges. In practice it usually means when $dist(m(p_i), p_i) \leq \lambda$, for some predetermined, typically very small λ; or until a maximum number of iteration T_{max} is reached. The convergence is then regarded as the mode of the underlying distribution. Finally, the points which share the same mode (or approximately close) is put in the same cluster.

12.3.2 DBSCAN

Density-Based Spatial Clustering of Applications with Noise (DBSCAN) [17] is a density-based clustering algorithm. DBSCAN was chosen for two main reasons. Firstly, it is one of the most popular clustering algorithms and highly cited in scientific literature. Secondly and more importantly, DBSCAN only requires a distance parameter. This parameter is known as ε or the radius of the circle surrounding each data point. Additionally, a secondary parameter *minPts* - the minimum number of neighbours, is required. *minPts* is optional and if unspecified it can be derived from the data set.

We briefly describe the algorithm as follows [62]. The said parameters are used for classifying points into three categories: *core*, *non-core* and *noise*, as follows. Consider a point p and a circle C with radius ε, centred at p. The point p is classified as a *core* point if and only if there are at least *minPts* number of *neighbours* points within C. These points are said to be *directly reachable* from p. A point q is reachable from p if there exists a path from p to q formed by $p_1, \dots, p_n$ where $p_1 = p$

and $p_n = q$, in which p_i for $1 < i < n$ must be a core point in its own disc. The only exception is the final point q. All points not reachable are *noise*.

Upon inspection, if a point p is a core point, then it forms a cluster together with all other points that are reachable from it. The *non-core* points can be part of a cluster, however they are the *boundary* points and cannot be used to reach further points.

Based on these definitions, the algorithm proceeds as follows. Start with a random point p that has not been visited and retrieve its neighbour set $N(p)$. Mark p as *visited*. If $|N(p)| < minPts$, then p is labelled as *noise* and another random point is selected to continue. Else, a new cluster C_p is started. It is then expanded by repeatedly exploring each *unvisited* neighbour $p_i \in N(p)$. Subsequently if $N(p_i)$ is large enough i.e. $|N(p_i)| \geq minPts$, these neighbours are also added to C_p. Otherwise, p_i is considered as a boundary point of C_p. For any point q which has been labelled as *noise* (hence not yet part of any cluster) but later found in a sufficient dense neighbourhood i.e. $q \in N(p_j) : |N(p_j)| \geq minPts$, then q is added to that cluster as a boundary point. The algorithm stops when every vertex is visited.

12.3.3 k-MXT

To cluster spatial data using the k-MXT algorithm, first we generate a *geometric* graph as follows. Given the set of points to be clustered, for each point v we add an edge (v,u) to a neighbour vertex u if and only if

$$dist(v,u) \leq d$$

where $dist(v,u)$ is a distance function and d is a specified constant. In the end, a *disc-graph* $G(V,E)$ is generated, which is used as the input for our k-MXT algorithm.

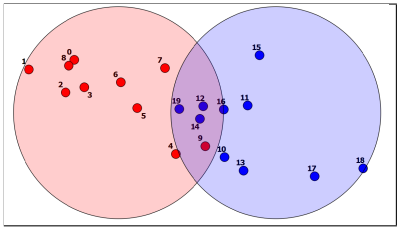
12.3.4 Example

Consider an example in which we generate, in total, 20 points randomly over two unit discs, both has radius 100 and the distance between the centres is 150. And suppose we apply the algorithms with the following parameters:

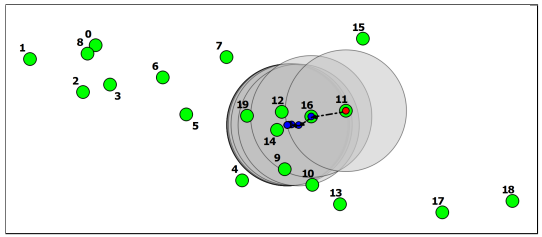
- k-MXT: $d = 40$, $k = 1$;
- DBSCAN: $\varepsilon = 40$, $minPts = 3$;
- Mean shift: $h = 40$, $T_{max} = 50$, $\lambda = 10^{-4}$.

For the example, we denote each vertex/data-point by its index/label as in Figure 12.5. Now suppose vertex 11 is selected, Figure 12.5 (a), (b) and (c) illustrate the procedure involving vertex 11 for mean shift, DBSCAN and k-MXT, respectively. Subsequently, Figure 12.5 (d) (e) (f) show the clustering results for each corresponding algorithm. To visualise the clusters, vertices in the same cluster are drawn using the same colour. Additionally, a convex hull is drawn (*dash-dotted* line) to cover each cluster where applicable (cluster size is greater than 2). For k-MXT result (Figure 12.5(f)) the *solid lines* are added to illustrate the resulting out-edges.

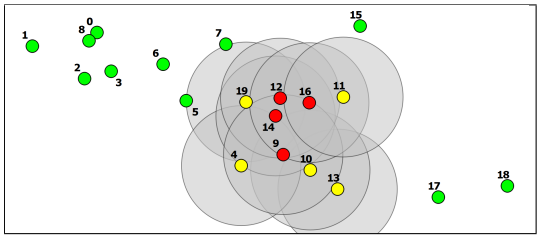
The algorithm procedures on the example.



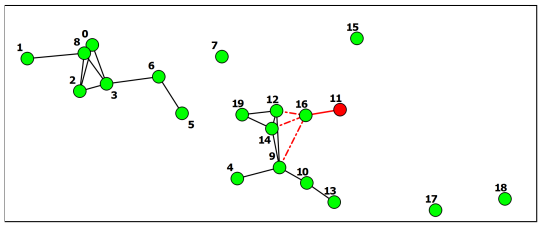
(a) Input



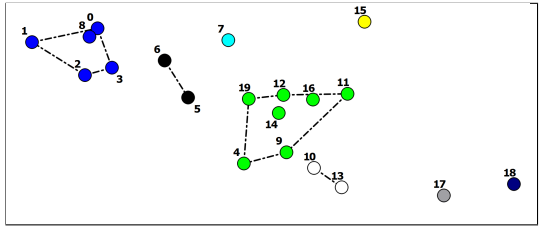
(b) Mean shift: the small *red dot* at vertex 11 denote the starting point. The blue points illustrate the trajectory of the shifted means which start from vertex 11, then shift to vertex 16, thereafter continue shifting toward *centre of mass* near the vertices $12, 14$, etc. Eventually, it converges near vertex 14 and terminates.



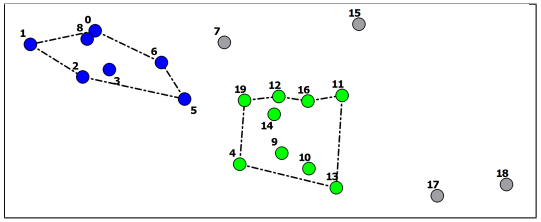
(c) DBSCAN: initially, vertex 11 is selected and classified as a *noise* point, since it has only one neighbour 16. Later on, vertex 16 is queried; since it is a core (red), 11 is then re-classified as a boundary point (yellow) and added to its cluster. Vertex $12, 14, 9$ are eventually added to this cluster as *core points*; and subsequently the remaining boundary points $4, 19$,etc. are added since they are reachable from these cores.



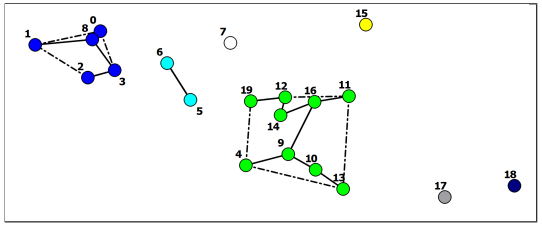
(d) k-MXT: since vertex 11 has degree 1, the edge $(11, 16)$ is chosen by default. Next, consider vertex 16, the edges $(16, 12)$, $(16, 9)$ and $(16, 14)$ all have weight 2, thus one is chosen uniformly at random, say $(16, 14)$. Furthermore, since $w(12, 14) = 3$, these two vertices select each other i.e edge $(12, 14)$ and $(14, 12)$ are added. And so on.



(e) Mean shift: result



(f) DBSCAN: result



(g) k-MXT: result

Fig. 12.5 The algorithms operations and results.

12.4 Experiment setting

12.4.1 Datasets

We use two datasets. The *small set* consisting of $4,000$ data points within a bounding rectangle covering St. Paul's Cathedral and Tate Modern (see Figure 12.7-a), extracted from the larger dataset. The *large set* consisting of $45,000$ data points covering a larger region of central London (see Figure 12.2).

12.4.2 Algorithm parameters

To calculate distance, we use the latitude-longitude coordinates and treat them as points in a plane, because the errors are tolerable given the scale (distance) we are looking at. For each algorithm, we consider three *disc radiuses*: 10 metres, 25 metres and 50 metres. Additionally, we list the other parameters used for each algorithm as follows

- k-MXT: $k = 1,2,3,4$;
- DBSCAN: $minPts = 3,20,40,80$;
- Mean shift: $T_{max} = 50$; $\lambda = 10^{-5}$.

Note that, for k-MXT, the larger k gets, the larger the size of each component. On the contrary, for DBSCAN, the larger $minPts$ gets, the smaller the size of each component. This is because, larger value of $minPts$ means the higher number of points is required to be found within any point's ε-neighbourhood for that point to be considered a core point. Therefore, as $minPts$ increases, more points would be labelled as *noise*, and therefore the clusters' sizes are reduced.

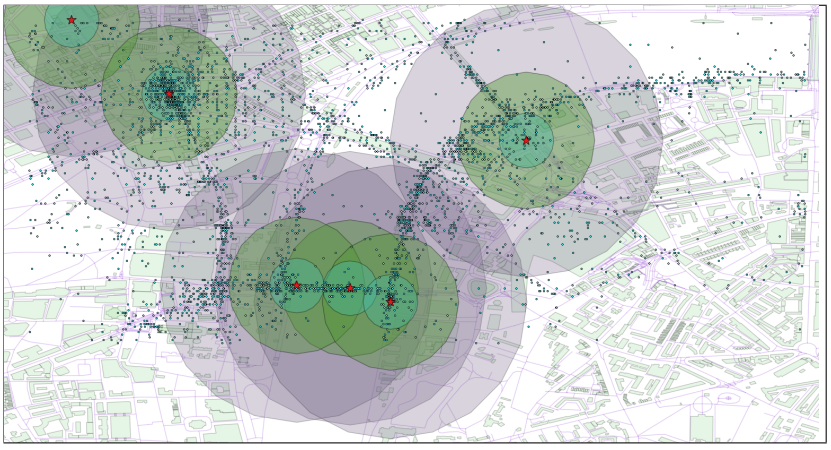


Fig. 12.6 The discs on each point. Each centre is marked as a **red star**. The radius of the corresponding disc is 10m, 25m and 50m for the small (turquoise), medium (green) and large (purple) disc, respectively.

12.4.3 Visualisation of results

To visualise the clusters produced by the algorithms, for each cluster we draw a *convex hull* which is the minimum bounding polygon covering every point within that cluster. Furthermore, to distinguish the clusters generated by different parameters, we use different colours for the bounding polygons. Particularly, the colour coding schemes in Table 12.1 is applied to the polygons.

Additionally to improve the visual aspect of k-MXT, we performed two *cleaning* steps to remove *small, noisy* clusters. Firstly, we removed clusters which have size less than a preset value s, which is typically the average size. These can be considered as *noises* in the dataset. Secondly, we removed the polygons lie completely within a larger polygon. It should be noted that the overlapping polygons is due to the nature of *convex hull* construction of each cluster. Some alternatives have been considered i.e. *alpha-shape* which yield little improvement but also introduce some further problems, hence it is excluded in the end.

To see the *growth or expansion* of clusters, as affected by increasing/decreasing the parameter $k/minPts$, we *overlayed* the resulting clusters. For instance, Figure 12.7 (c) (d) (e) and (f) present results produced by k-MXT, corresponding to $k = 1, 2, 3$ and 4, respectively. These figures are *overlayed* to produce one single *main figure* Figure 12.7 (b). This step is done for every algorithm.

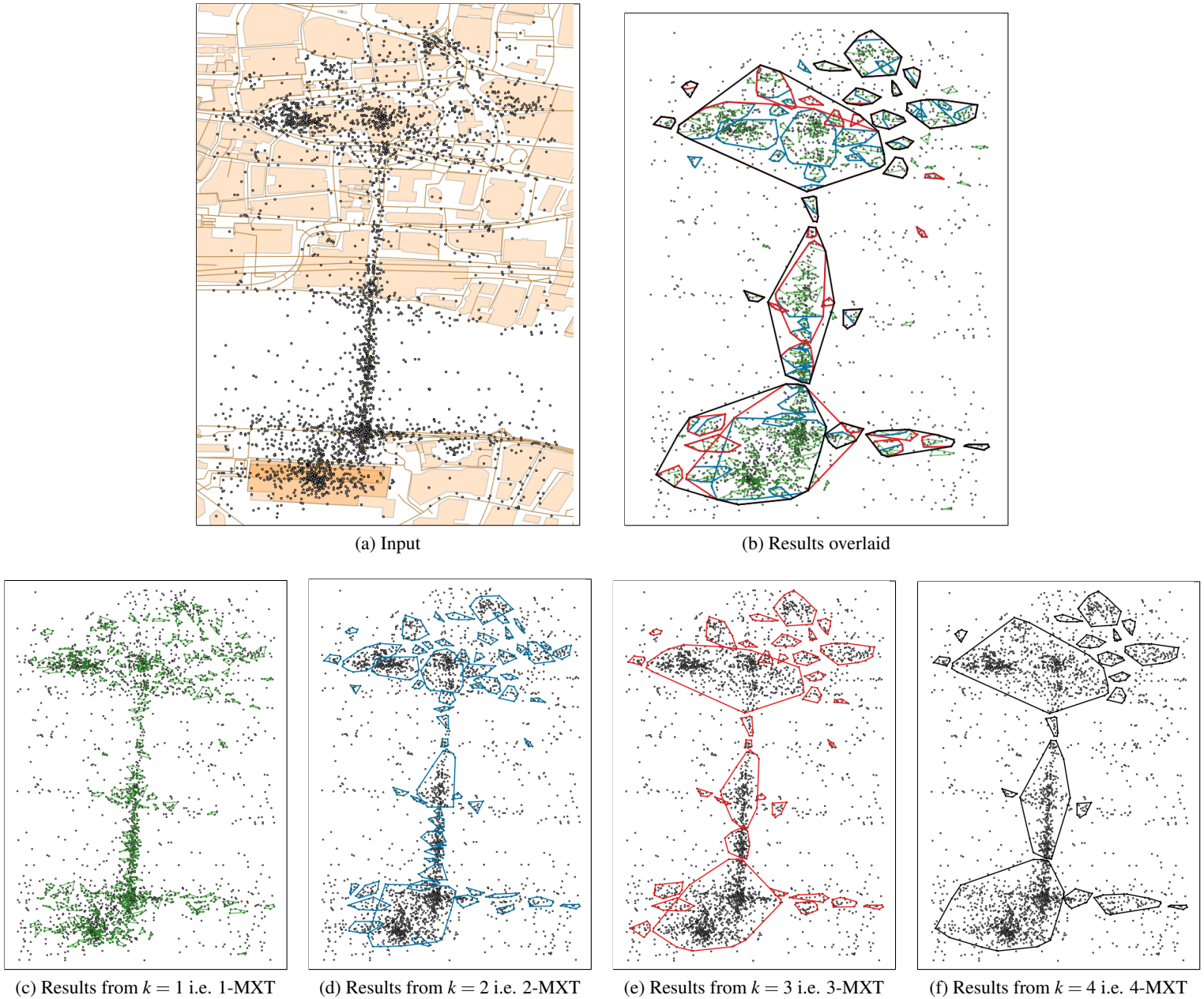
Based on initial experimental results, we divided the algorithms' parameters into groups corresponding to the size of the resulting cluster, as shown in Table 12.1 below. In the following sections of this chapter, we continue to use this *classification* to compare and evaluate the algorithms' results.

	Corresponding parameter			Colour	
Cluster size	Mean shift	minPts (DBSCAN)	k (k-MXT)		
Large	50m	3pt	4	●	Black
Medium	25m	20pt	3	●	Red
Small	10m	40pt	2	●	Blue
Very small		80pt	1	●	Green

Table 12.1 Colour coding schemes correspond to specific parameter and also the size of the resulting cluster.

12.5 Results

k-MXT applied to the small dataset $d = 10$m.

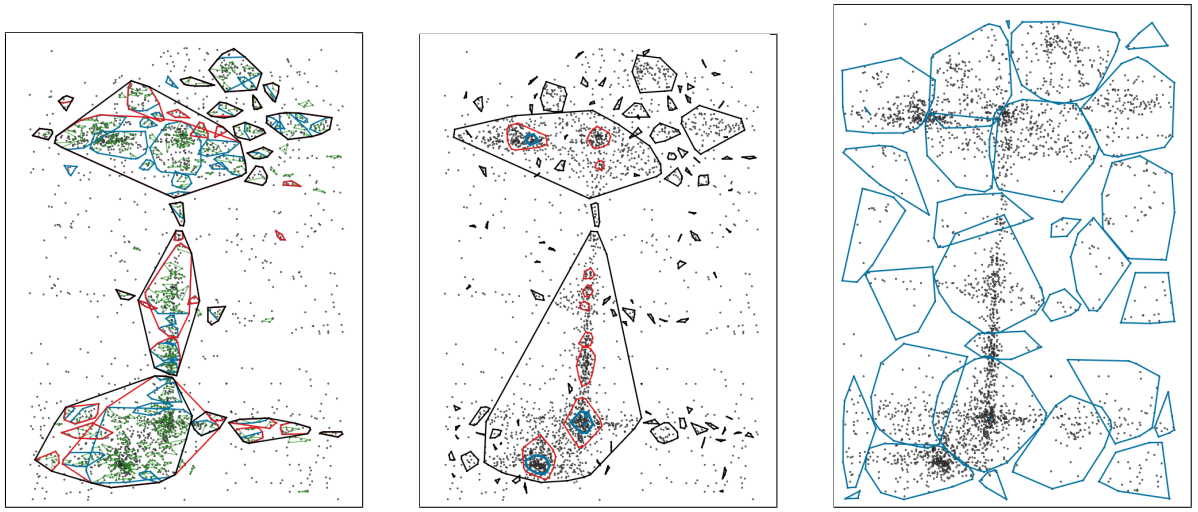


(a) Input (b) Results overlaid

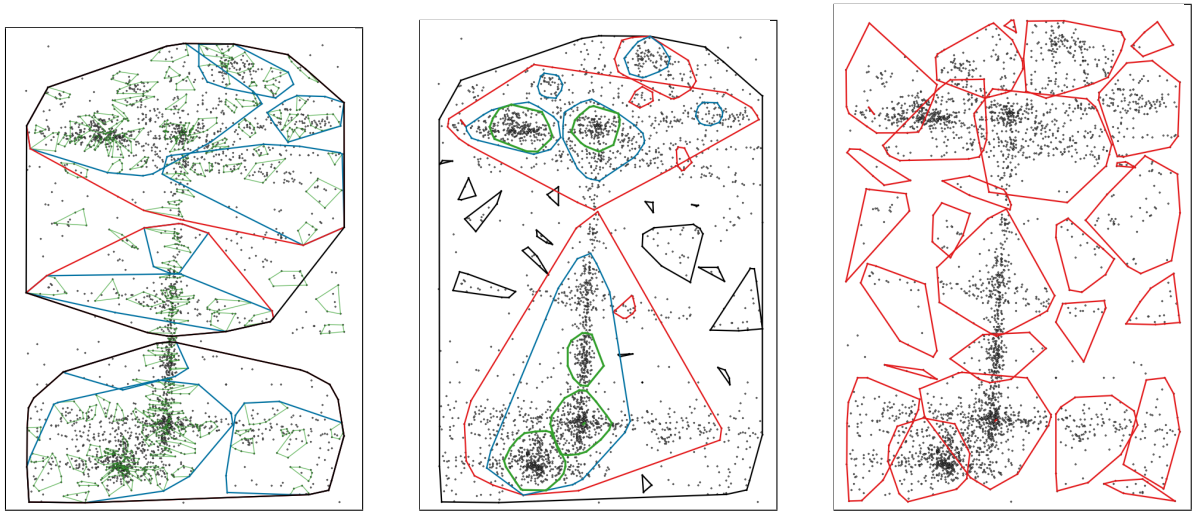
(c) Results from $k = 1$ i.e. 1-MXT (d) Results from $k = 2$ i.e. 2-MXT (e) Results from $k = 3$ i.e. 3-MXT (f) Results from $k = 4$ i.e. 4-MXT

Fig. 12.7 k-MXT algorithms on the *small dataset* with $d = 10$m and $k = 1, 2, 3$ and 4. For $k \geq 2$, some regions are clearly defined. One can roughly identify some straight lines which are probably bridges, roads or footpaths. Notice that the *polygons* expand with k.

Results of small dataset.



(a) k-MXT (b) DBSCAN (c) Mean-shift



(d) k-MXT (e) DBSCAN (f) Mean-shift

Fig. 12.8 **Top figures:** $d, \varepsilon, h = 10m$. **Figures below:** $d, \varepsilon, h = 25m$.

k-MXT on large dataset, $d = 10$m.

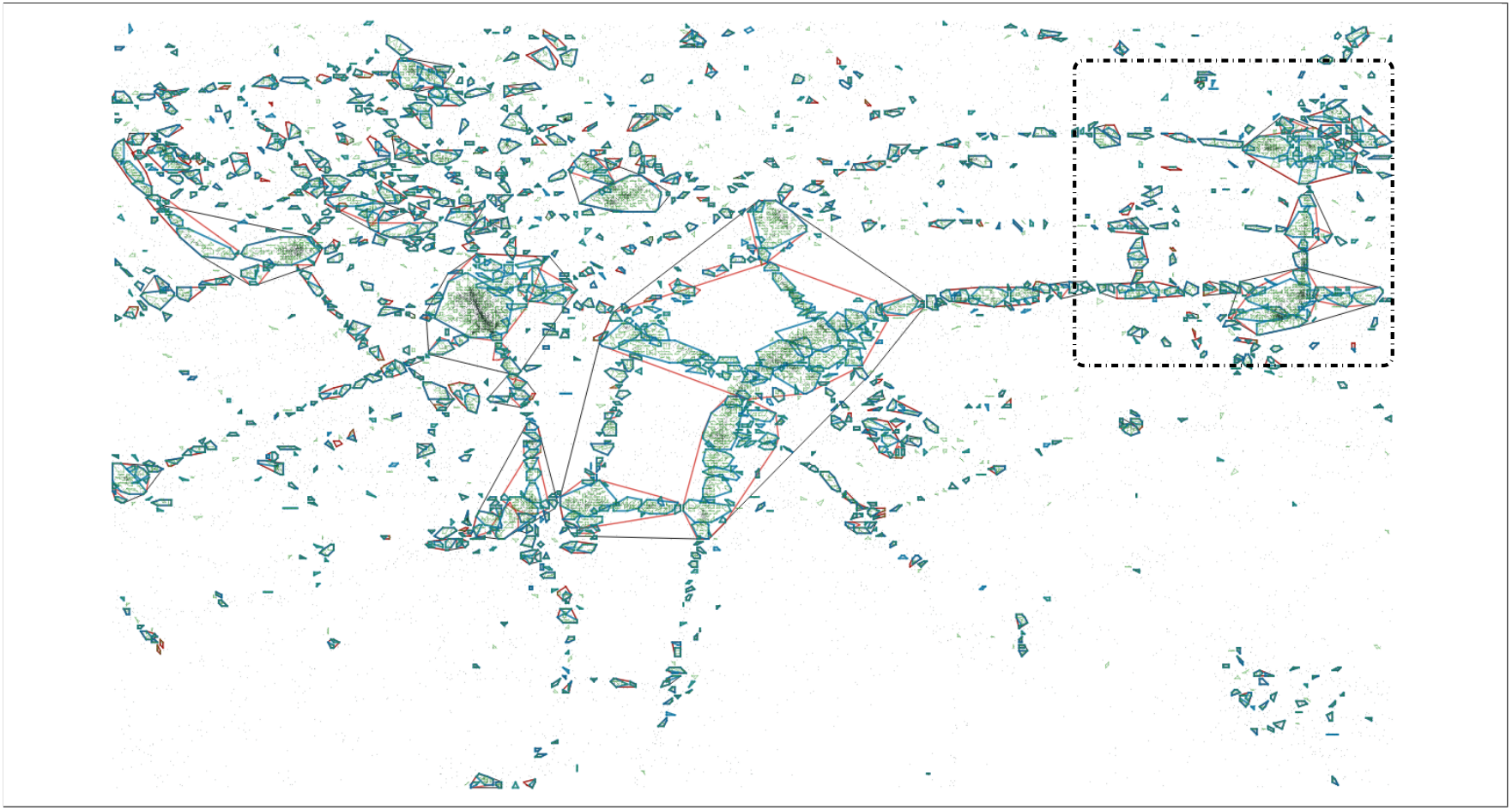


Fig. 12.9 k-MXT($d = 10$)m. The results for each k have been overlaid on the same layer, demonstrating the expanding of region/cluster as k is increased. The region covered by the *dash-dotted rectangle* is used as the **small dataset**, shown in previous Figure 12.7 and 12.8.

DBSCAN on large dataset, $\varepsilon = 10$m.

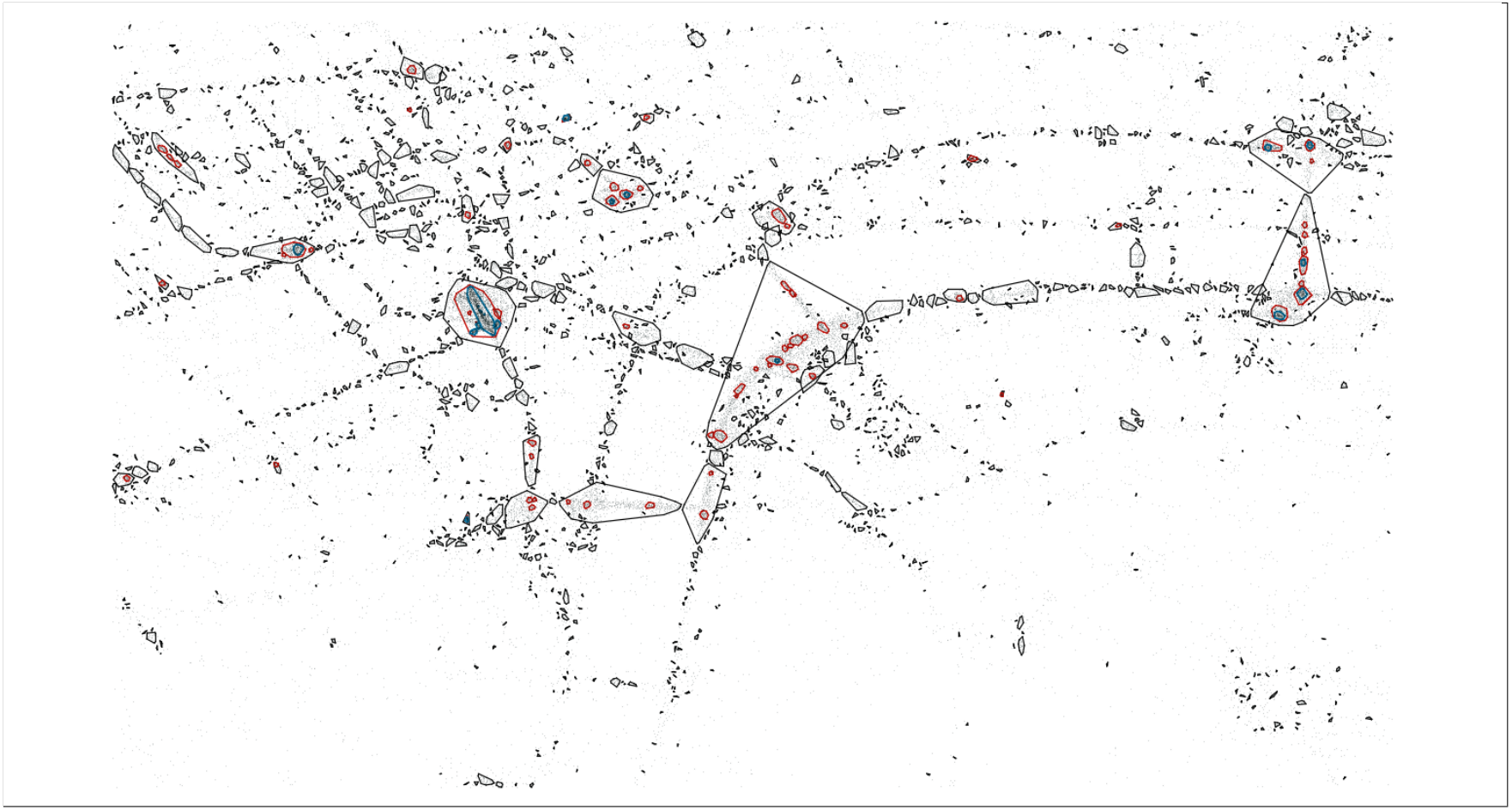


Fig. 12.10 DBSCAN($\varepsilon = 10m$). As *minPts* is increased, the number of clusters is decreased as the area of the polygons also decrease. There is no visible green (correspond to $minPts = 80$) polygon, few of the blue poylgons ($minPts = 40$), while the area of the red polygon ($minPts = 20$) are relatively small.

k-MXT on large dataset, $d = 25$m.

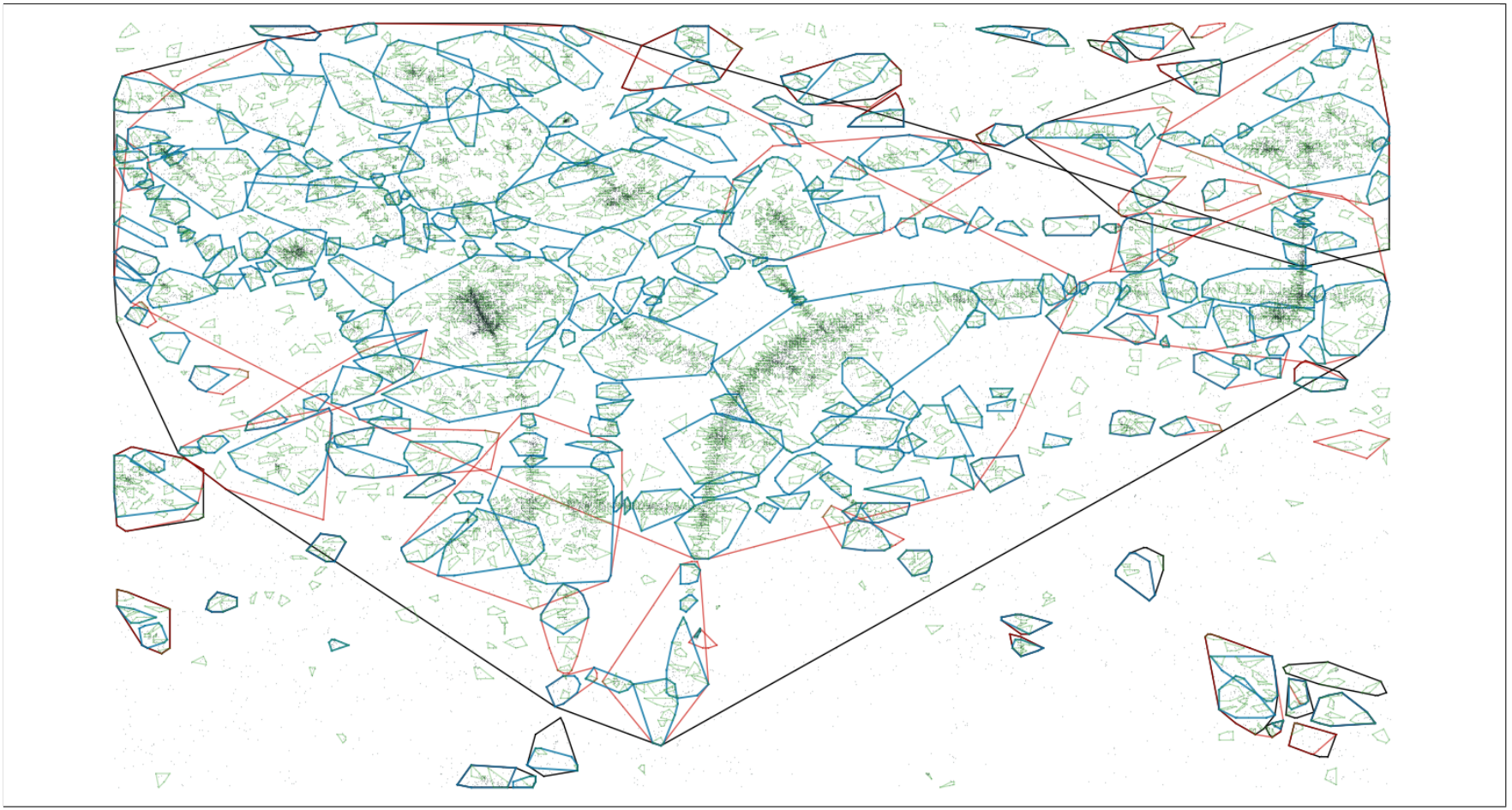


Fig. 12.11 k-MXT($d = 25m$). The green (smallest, $k = 1$) polygons are very small and its orientation seems quite arbitrary. The blue ($k = 2$) polygons have expanded considerably but still retain some interests. While the red ($k = 3$) and black ($k = 4$) regions cover very large areas of central London.

DBSCAN on large dataset, $\varepsilon = 25$m.

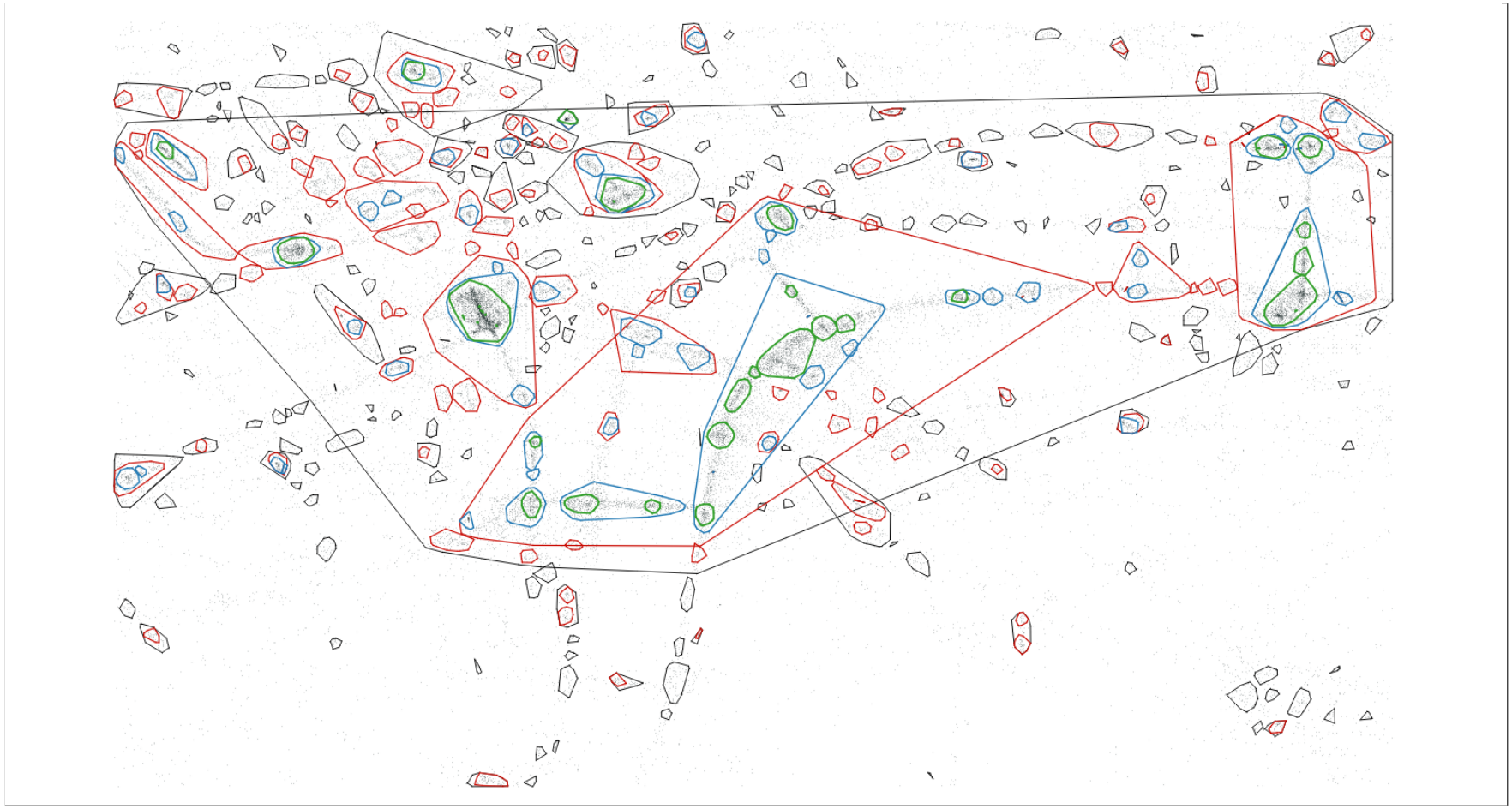


Fig. 12.12]

DBSCAN($\varepsilon = 25m$). Overall, the picture shows a decent level of clarity. Compare to k-MXT in figure 12.11, the areas of regions produced are much smaller. Notably, the green (small, $minPts = 80$) and red (medium, $minPts = 40$) polygons cover some interesting regions.

Clusterings produced by 2-MXT($d = 10m$), shown on map.

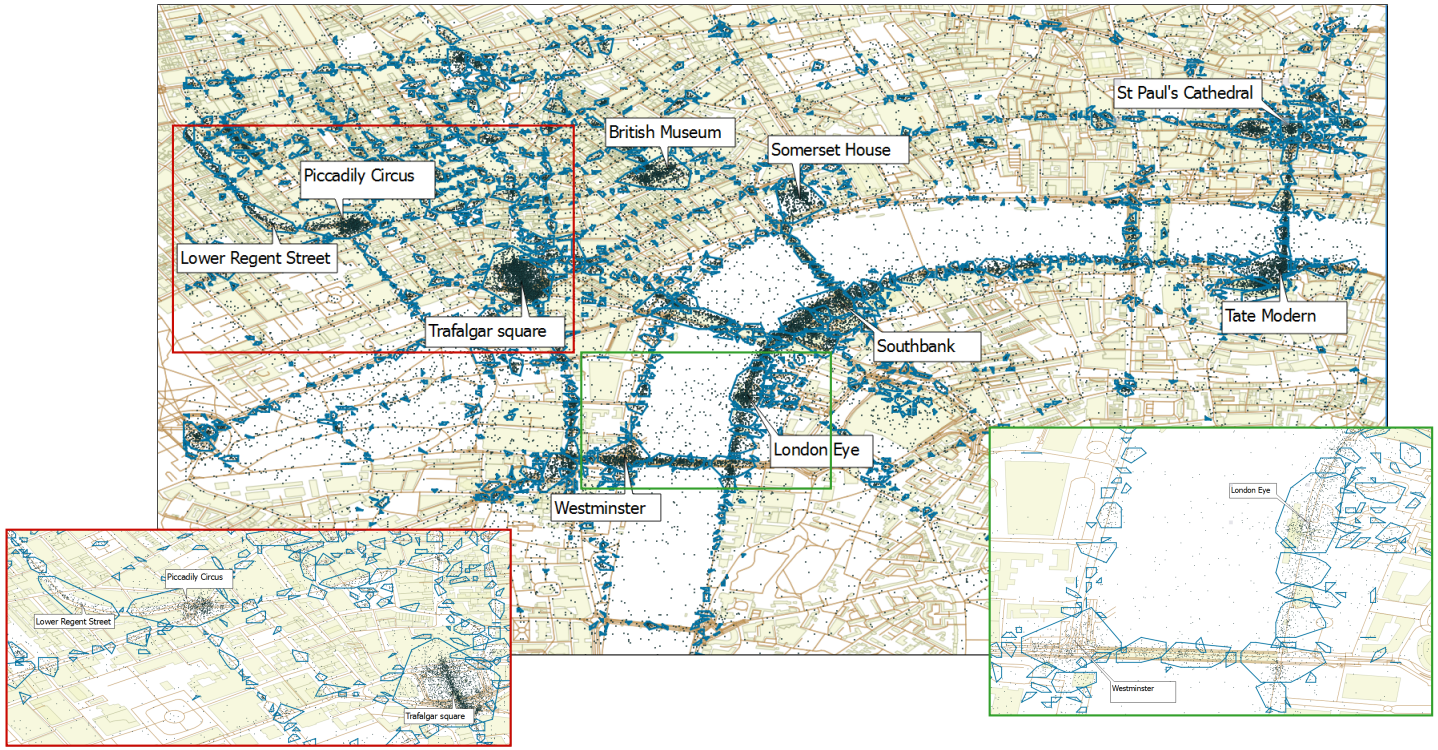


Fig. 12.13 2-MXT($d = 10$) metres, layered on London's map. Some popular tourist attractions are identified and labelled. It can be seen that there are clusters/polygons corresponding to these places. The two separated pictures provide a closer look at the two regions, specified by its border's colour.

12.6 Evaluation

In this section, we evaluate the performance of the algorithms in terms of accuracy and complexity.

12.6.1 Accuracy

Arguably, we lack a common quantitative measure of the classification accuracy for this problem. Hence, we evaluate the accuracy of the algorithms by visual inspection. More specifically, we inspect whether the polygons produced by the algorithms cover *meaningful* regions (see Figure 12.13 for a map of London with some identified points of interest).

For the *small dataset*, we can see in Figure 12.8 that each algorithm is able to produce clusters which cover some dense regions of dots, which correspond to the landmarks: Tate Modern Gallery, Millennium Bridge and St Paul's Cathedral. The polygons produced by mean shift seem similar regardless of the distance h i.e. Figure 12.8 (c) and (f). On the other hand, DBSCAN and k-MXT produce quite distinct clusters. Among the algorithms, DBSCAN produces the best results i.e. Figure (b) and (e), being able to *closely fit* the regions of the said attractions.

For the *large dataset*, Figure 12.9 and 12.10 show the clusters produced by k-MXT and DBSCAN with $d, \varepsilon = 10$ metres. Overall, both results look comparable. Some of $2,3,4$-MXT and DBSCAN($minPts = 3$) polygons appear to cover some popular tourist attractions like Trafalgar Square, Westminster, British Museum etc. For DBSCAN with larger values of $minPts$, the polygons become significant smaller hence less visible.

Figure 12.11 and 12.12 present results for $d, \varepsilon = 25m$. Some polygons produced by 2-MXT (blue) are recognisable i.e. Trafalgar square and Piccadilly Circus. The remaining clusters are rather large, covering *much wider regions* which happen to contain many landmarks; thus considered a loss of accuracy. On the other hand, DBSCAN with $minPts = 40, 80$ produce clear and identifiable regions of interests.

To improve the performance of k-MXT we can introduce a second parameter w the minimum weight of an edge. This is equivalent to the $minPts$ parameter of DBSCAN. See section 12.7 for details. Hereafter, we refer to this algorithm as k-MXT(d, w).

For further test of effectivity, we perform some *blind tests*. In which, we randomly select a city which we are not familiar with, collect data then cluster it. Finally, we examine the resulting clusters for potential *meaning*. See section 12.8 for results.

12.6.2 Complexity

The overall procedure of k-MXT can be broken down into two main tasks: the graph *construction* and the *k-MXT fragmenting*. We describe a simple version as follows.

12.6.2.1 Graph construction.

To construct the *disc graph*, we need to find the vertices inside each disc. The naive approach is as follows. For each vertex v, we check each vertex u and see if u is in the disc centred at v i.e. $dist(v,u) \leq d$. This takes $O(n^2)$ operations.

To improve this, we can reduce the number of vertices to be examined as follows. We separate the x and y coordinates into two arrays. More specifically, suppose each vertex $v = 1,2,\ldots,n$ has its coordinates as (x_v, y_v). We then store the coordinates in two arrays

$$X = \{x_i : i = 1,2,\ldots n\} \quad \text{and} \quad Y = \{y_j : j = 1,2,\ldots,n\},$$

X and Y are then sorted. This operation takes $O(n \log n)$.

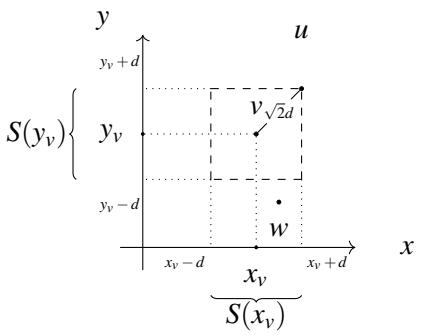


Fig. 12.14 An example demonstrates the *spatial search* procedure. Assume the queried vertex is $v(x_v, y_v)$ and there are two vertices $u(x_u, y_u)$, $w(x_w, y_w)$. Then, $S(x_v) = \{u, w\}$ and $S(y_v) = \{u\}$. Thus the intersection of these two sets gives $S(x_v) \cap S(y_v) = \{u\}$. Hence, we find that u is located within the *bounding square* centred at v.

For each vertex $v = (x_v, y_v)$ and each coordinate x_v and y_v, we locate the values within a fixed distance from it using range binary search. The result, for *each coordinate*, is a set of points which are located within a fixed distance d from the queried point i.e.

$$S(x_v) = \{u : |x_u - x_v| \leq d\} \qquad S(y_v) = \{w : |y_w - y_v| \leq d\}. \tag{12.3}$$

As the coordinate arrays are sorted, a search operation on the x and y co-ordinate takes $O(\log n)$ for each vertex; hence $O(n \log n)$ overall.

Given $S(x_v)$ and $S(y_v)$, we want to find the set of vertices of which the distances to v is at most d. It is seen that the intersection of these two sets i.e. $S(x_v) \cap S(y_v)$ yields the set of points bounded by a square with width d centred at v, see for example Figure 12.14. To transform the *bounding square* to a *bounding circle* then requires an additional computation step, in addition to the set intersection. Alternatively, since

$$|S(x_v) \cap S(y_v)| \leq \min\{S(x_v), S(y_v)\},$$

we can take the smaller set, say $S(x_v)$, then for each $u \in S(x_v)$ connect (v,u) iff $dist(v,u) \leq D$. Furthermore, it is seen that either $S(x_v)$ or $S(y_v)$ is required to complete the search procedure. Therefore, this final operation takes $O(n \times \max_{v \in V}\{|S(x_v)| \text{ or } |S(y_v)|\})$.

Overall, the complexity of the graph construction takes

$$\underbrace{O(n \log n)}_{\text{sort}} + \underbrace{O(n \log n)}_{\text{range search}} + \underbrace{O(n \times \max_{v \in V}\{|S(x_v)| \text{ or } |S(y_v)|\})}_{\text{connect edges}} = O(n^2),$$

since $\min\{S(x_v), S(y_v)\} \leq n$ and it is difficult to get an exact bound for the this term. The size of $S(x_v)$ or $S(y_v)$ is close to n is close in extreme cases, for example, points are distributed like a *cross* shape; or the distance d is set to the maximum between any pair of vertices. Nevertheless, such cases are rare and d is typically very small. For example, in our experiments with $d = 10$ metres, the sets $S(x_v)$, $S(y_v)$ are, at most $\approx 2,000$ points, much smaller compares to $n \approx 45,000$.

If the data is geo-tagged data at world scale there is a further consideration in the calculation of (12.3). That is, using the *improved-naive* method, we can also improve the running time when calculating using geographical coordinates i.e. latitude/longitude. The problem here is that calculating the Earth or great-circle distance involves trigonometry which further complicates the computation. Therefore the motivation is to reduce the number of necessary great-circle distance computations, using the same method as above.

More specifically, we construct the disk graph in two steps. *The first step* is to find the *bounding square* as in (12.3). This is done using a parameter d' which is treated as the Euclidean approximation. This would *overestimate* the actual earth distance, however *can be computed faster*. *The second step* is to find points within the *bounding disk* given the bounding square found in step 1. This is done using parameter d, which treated as the *great-circle distance* computed accurately using e.g. the Haversine formula. Essentially, in the first step, we traded precision for speed by using Euclidean; and only compute the precise distance in the second step.

In experiments, we relate d' to d using $d' = d \times 10^{-2} + \varepsilon$. For instance, to construct the disk graph with radius of 10 metres, we set $d = 0.01$ (since Haversine returns distance in kilometre) and $d' = 0.00015$. The *bounding squares* produced by the d' parameter typically have width ≈ 30 metres. A sanity check is done by comparing the resulting number of edges with that produced by the naive method.

Further improvement requires using spatial data structures i.e. *R*-tree or *kd*-tree. These are *spatial trees*, specialised for indexing spatial data. Using these to find neighbours within a *rectangle* is improved. The construction of such trees take, on average, $O(n \log n)$. A search query takes $O(\log n)$ on average and $O(n)$ worst. Also, note that both structures' *search operation* only supports query by *rectangle*. Thus, an additional step is required to locate points within a vertex's *circle*.

We use the C++ Boost Geometry library [57] for an implementation of *R*-tree and the Approximate Nearest Neighbour (ANN) [11] for *kd*-tree. Since it is possible to index geographical coordinates,

or spherical coordinates assuming the Earth is a perfect sphere, with the *R*-tree, we also include the running time for graph generation using the great circle distance for comparison.

Table 12.2 presents the running times in seconds for graph construction on the large dataset which consists of approximately $45,000$ data points. The *improved naive* method works surprisingly well, its performance is comparable to that of the spatial trees'; and even slightly better when dealing with spherical distance compares to the R-tree.

	Naive		Improved Naive		*R*-tree		*kd*-tree
	Cartesian	Spherical	Cartesian	Spherical	Cartesian	Spherical	Cartesian
10m	82.1	419.4	1.6	**5.6**	1.3	9.1	**1**
25m	82.1	419.4	3.6	**13.59**	3.1	26.7	**1.5**
50m	82.1	419.1	6.9	**26.35**	7.3	65.3	**2.6**

Table 12.2 Large dataset: $n = 45,000$. Graph construction time in *seconds* and averaged over 10 executions. The best running time are highlighted. Interestingly, the *improved naive* method performs better than the *R*-tree when computing the *spherical* coordinates. Which perhaps hints at a limitation of the data structure on curved spaces. The naive timings are the same, as all data is processed in each case.

12.6.2.2 Fragmenting using k-MXT

The fragmenting process consists of the following tasks:

1. For each edge: calculate the number of common neighbours i.e. its weight;
2. For each vertex: select k highest scores;
3. Find the connected components of the resulting graph fragments.

The *first task* is equivalent to finding the set intersection of two given adjacency lists $A(v), A(u)$ on each edge (v,u). Suppose vertices are indexed i.e. $v = 1,2,\ldots,n$ then each adjacency list is an array of integers. Thus, naive set intersection on any edge (v,u) takes $O(d(u) \times d(v))$. Hence overall it takes $O(|E| \times d_{max}^2)$ where d_{max} is the maximum degree in the graph i.e.

$$d_{max} = \max\left\{d(v) : v \in V\right\}.$$

An improvement to this task can be achieved if the adjacency lists are sorted. This implies a $O(n \times d_{max} \log d_{max})$ of *pre-processing* i.e. sorting time. Then the computational cost for each set intersection operation on an edge (v,u) takes at most $d(u)+d(v)$. This is done by scanning both A(v), A(u) simultaneously i.e. start by having two pointers at $A(v)[0]$ and $A(u)[0]$, increment the pointer of the former if $A(v)[0] < A(u)[0]$ and vice versa, or both if the two are equal and so on. Hence, the overall computation cost is

$$\begin{aligned}\sum_{v\in V}\sum_{u\in A(v)}\big(d(v)+d(u)\big) &= \sum_{v\in V} d^2(v) + \sum_{u\in A(v)}\sum_{v\in V} d(u) \\ &= 2\sum_{v\in V} d^2(v) \leq 2d_{max}\sum_{v\in V} d(v) = 4|E|d_{max}.\end{aligned}$$

Thus the task takes $O(|E| \times d_{max})$. It is seen that the complexity of this procedure depends on d_{max}. Hence, for graphs with low d_{max} this procedure would perform well. However, for graphs with skewed degree distribution e.g. power law, d_{max} can be large thus resulting in a slower running time. In which case, since this problem is analogous to the problem of identifying triangles in graphs, a better triangle counting algorithm can be considered. For instance, we considered the *forward* algorithm by [52] which has a complexity of $O(|E|^{3/2})$ (see Section 12.6, at the end of this chapter). On our *large dataset* with $d = 50m$, we have $d_{max} \approx 1500$. Then the *forward algorithm* takes 80 seconds to finish while the implemented procedure takes 62 seconds. Nevertheless, in much larger and denser dataset, the *forward method* from [52] may help to improve the running time.

The *second task* is done using a priority queue i.e. min-heap. First, set the size of the heap to k. Then, iterate through each adjacency list, a vertex is enqueued if its number of common neighbours is greater than the current root's; else if draw then e.g. select one at random. Thus, the heap contains the k largest elements with the root being the smallest. For each insertion, to restructure the heaps takes $O(\log k)$. Hence for each vertex v it takes at most $d(v)\log k$. Therefore, the task takes at most $\sum_{v \in V} d(v)\log k = \log k \times 2|E| = O(|E|)$, for fixed k.

The *final task* is to compute the graph's connected components. This can be done using any classical algorithm in linear time in the number of edges in the component i.e using breadth-first search or depth-first search. With small values of k the resulting fragments are relatively sparse therefore this task can be done relatively fast e.g. using disjoint-set data structure which runs in linear time of the number of edges hence $O(kn) = O(n)$.

Overall, the *fragmentating process* has a running time of

$$\underbrace{O(n \times d_{\max} \log d_{\max})}_{\text{pre-processing}} + \underbrace{O(|E| d_{\max})}_{\text{set intersects}} + \underbrace{O(|E|)}_{k\text{-max scores}} + \underbrace{O(n)}_{\text{connected components}} .$$

If we *ignore the one-off pre-processing*, then the complexity is $O(|E| d_{\max})$. In comparison, DBSCAN implemented with *R*-tree or *kd*-tree has an average complexity of $O(n \log n)$ [17]. This is because in general case that the ε-neighbourhood is typically much smaller than the whole dataset, then a region query takes $O(\log n)$ using the said data structures [17]. For mean shift, a loose theoretical running time is $O(n \times T_{\max})$ where $T_{\max}$ is the maximum number of iterations allowed for each query. In practice, we usually set the λ (i.e. the distance to determines convergence) at $0.5m$, and notice that the mode converges in relatively fewer iterations than T_{max}. DBSCAN is executed using the R package "dbscan" [40], a *fast reimplementation* of the original algorithm in C++. Mean shift is executed using the R package MeanShift [39].

Table 12.3 presents the overall running time of the algorithms. The results of the small experiments show the mean shift has the slowest running time; hence it was excluded in the large experiment. Furthermore, in both experiments, DBSCAN outperforms *the current implementation* of k-MXT. For k-MXT, it is seen that the running time for the *clustering procedure* seems to scale quadratically with

the distance, which determines the number of edges in the graph hence d_{max}. This is probably due to the $O(|E|d_{max}) \approx O(n(d_{max})^2)$ set intersection.

		k-MXT		DBSCAN	MeanShift
		$\lvert E\rvert$	Time		
Small Set (4,000 points)	10m	24,305	0.03	0.02	180
	25m	115,250	0.33	0.04	360
	50m	225,000	0.47	0.05	600
Large Set (45,000 points)	10m	300,000	0.56	0.38	
	25m	1,400,000	8.55	1.35	
	50m	4,400,000	61.5	4.12	

Table 12.3 Running time in *seconds*. Note: for k-MXT, *the pre-processing and graph construction time is excluded*.

12.6.3 A measure of density

A recurring issue when evaluating the performance of this type of problem is the lack of a meaningful measure. Therefore, we often resort to visual inspection in an attempt to evaluate goodness of the results. Which leads us to the question: What makes a cluster **look** better than the others?

Figure 12.15 shows six result clusters covering the region of Trafalgar square, taken from the large dataset. For which, we have the following polygon-algorithm correspondence

(a): 2-MXT($d = 25m$); (b): 2-MXT($d = 25m, w = 20$);

(c): DBSCAN($\varepsilon = 25, minPts = 40$); (d): 2-MXT($d = 25m, w = 40$);

(e): 3-MXT($d = 10m, w = 40$); (f): DBSCAN($\varepsilon = 10m, minPts = 80$).

So, which figure, visually, looks better compared to the other? The answer is, perhaps, (e) or (f). This is because, on the one hand, the top polygons i.e. (a) (b) and (c) cover exceedingly large areas. Thus there is a significantly amount of *empty space* in each figure. On the other hand, polygon (d) (e) and (f) looks much better, being able to *closely fit* the centre dense region. Although (f) is visibly the smallest, but it can be argued that it is too small as many of the relevant points lie outside the polygon. And hence, (e) is arguably *better* in this sense.

Motivated by this observation, we derive a reasonable measure of density as follows. Let $n(P)$ be the number of points within the polygon P, and $A(P)$ be its area, then the density $R(P)$ is defined by

$$R(P) = \frac{n(P)}{A(P)}. \tag{12.4}$$

Thus $R(P)$ measures the data density within a given region i.e. the number of contained data points per unit of area. In our experiments, $A(P)$ is typically measured in square metres m^2, thus $R(P)$ yields the number of dots per m^2. A high score indicates a high level of density of *dots* within such polygon. However, it may not always indicate a better *cluster*. We sometimes encounter clusters with

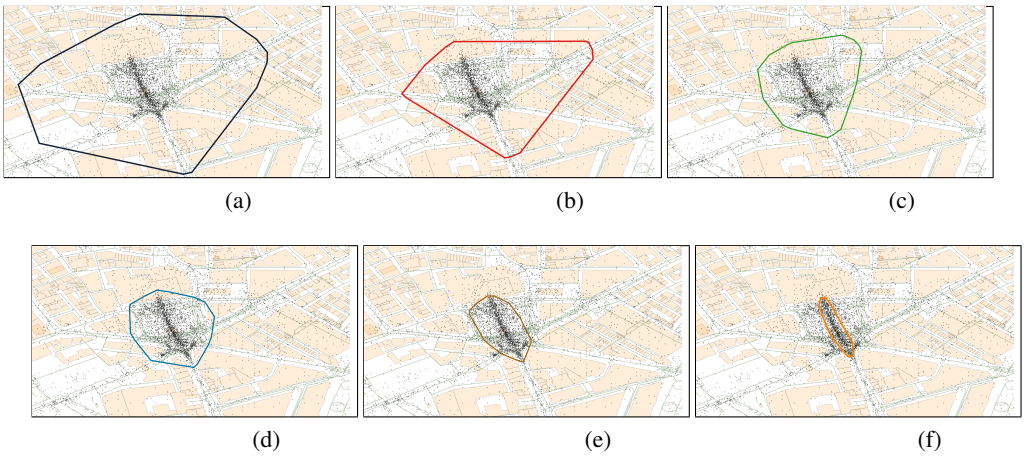


Fig. 12.15 Which figure *looks* better?

an abnormally high density e.g. those with few vertices in a tiny area ($< 1m^2$) thus yielding a very high density.

12.6.4 Natural scale of observation

For this type of spatial data, there is a natural distance parameter that associates with the scale of observation [10]. For example, looking at the *dots* at *global level* i.e. Figure 12.1, one can identify the continents and countries. At *city level* i.e. Figure 12.2 and 12.3, one can recognise a city's streets, landmarks, etc. Zooming closer to a *landmark level*, we are able to further identify *hot spots* within these landmarks i.e. Figure 12.7-(a): the Tate Modern Gallery and the head of Millennium Bridge are distinctive. Clearly, these scales of observation are not pairwise compatible, meaning one cannot distinguish hot spots in landmarks looking from the *city level* and vice versa.

Observing at *city level*, we found that the distances: 10m and 25m yield good clustering results. Furthermore, the convex hulls with areas between $1000 - 20,000m^2$ are often suitable for identifying locations of interest; outside this range the polygons are either invisibly small or exceedingly large. For instances, the areas of the polygons (d) (e) (f) in Figure 12.15 are $45,000$, $29,000$ and $4,700\ m^2$, respectively. Thus an area of few hundred square metres would probably look like a dot inside these polygons. In fact, Figure 12.16 shows an observation at *landmark level*, over the region of Tate Modern Art Gallery, examined in the small dataset in Figure 12.7. It is seen that identifiable polygon at this level has an area of between $200 - 1000m^2$, while the scale of observation was magnified three times. See Figure

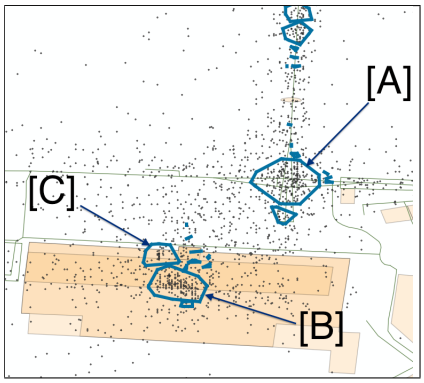


Fig. 12.16 Observation at *landmark level* over Tate Modern region. The areas of the polygon [A], [B] and [C] are 1000, 600 and 200 m^2, respectively.

12.8 top-right corner for an observation over this region at a *city level*.

Through experiments on our datasets, we found that a density between $0.1 \leq R(P) \leq 0.5$ can be considered a good result. For instances, the density of the polygons in *bottom row* in Figure 12.15 (d) (e) and (f) is 0.072, 0.101 and 0.291, respectively. Scores higher than 1 are often occur in very small polygons thus having an abnormally high score even though containing a small number of dots e.g. 5 *dots* in an area of $0.1m^2$ yields $R(P) = 50$. In some cases, this could be interesting for example many people taking pictures at the same location; however it could also mean a person taking lots of photo at the same spot, which is rather problematic.

Consider the large dataset consists of $45,000$ points over a region of $15km^2$ in central London i.e. see Figure 12.4. By (12.4) its reference density is

$$R(\text{large}) = \frac{45 \times 10^3}{15 \times 10^6} = 0.003.$$

Thus, one would expect a clustering to produce a better density than such. Similarly, for the small dataset the reference value is $R(\text{small}) = \frac{3500}{6.4 \times 10^5} = 0.0054$.

An interpretation of this dispersion of points is as follows. Suppose we divide this $15km^2$ region of central London into smaller squares of equal area. Let the result be a set S consists of $|S|$ number of squares with equal area s. Then, the average density of these squares is

$$\frac{1}{|S|} \sum_{S_i \in S} R(S_i) = \frac{1}{|S|} \sum_{S_i \in S} \frac{N(S_i)}{A(S_i)} = \frac{\sum_{S_i \in S} N(S_i)}{|S| \times s} = R.$$

This simple method of dividing the original bounding box into smaller squares of equal size can produce rather interesting results.

For instance, Figure 12.17 shows an example in which every square has an area of $2,500m^2$. Further, each square is coloured according to its density, from white (none - 0) to dark orange (densest - 0.2). The *large bounding box* then becomes a pixelated *picture*. For those who are familiar, the map of central London can be quite apparent. Furthermore, the *more photographed* areas are immediately recognisable by the *darker pixels*. However, a downside of this approach is that it is rather rigid i.e. it depends on the size of the unit square and the position of the minimum point (bottom-left) and maximum point (top-right). Also, there are lots of *empty space* in *sparser* squares e.g. see the smaller figure in Figure 12.17. Thus, a potential improvement of this heuristic would involve the removals of sparse squares to create space for expanding the *dense* squares in some directions to maximise its density.

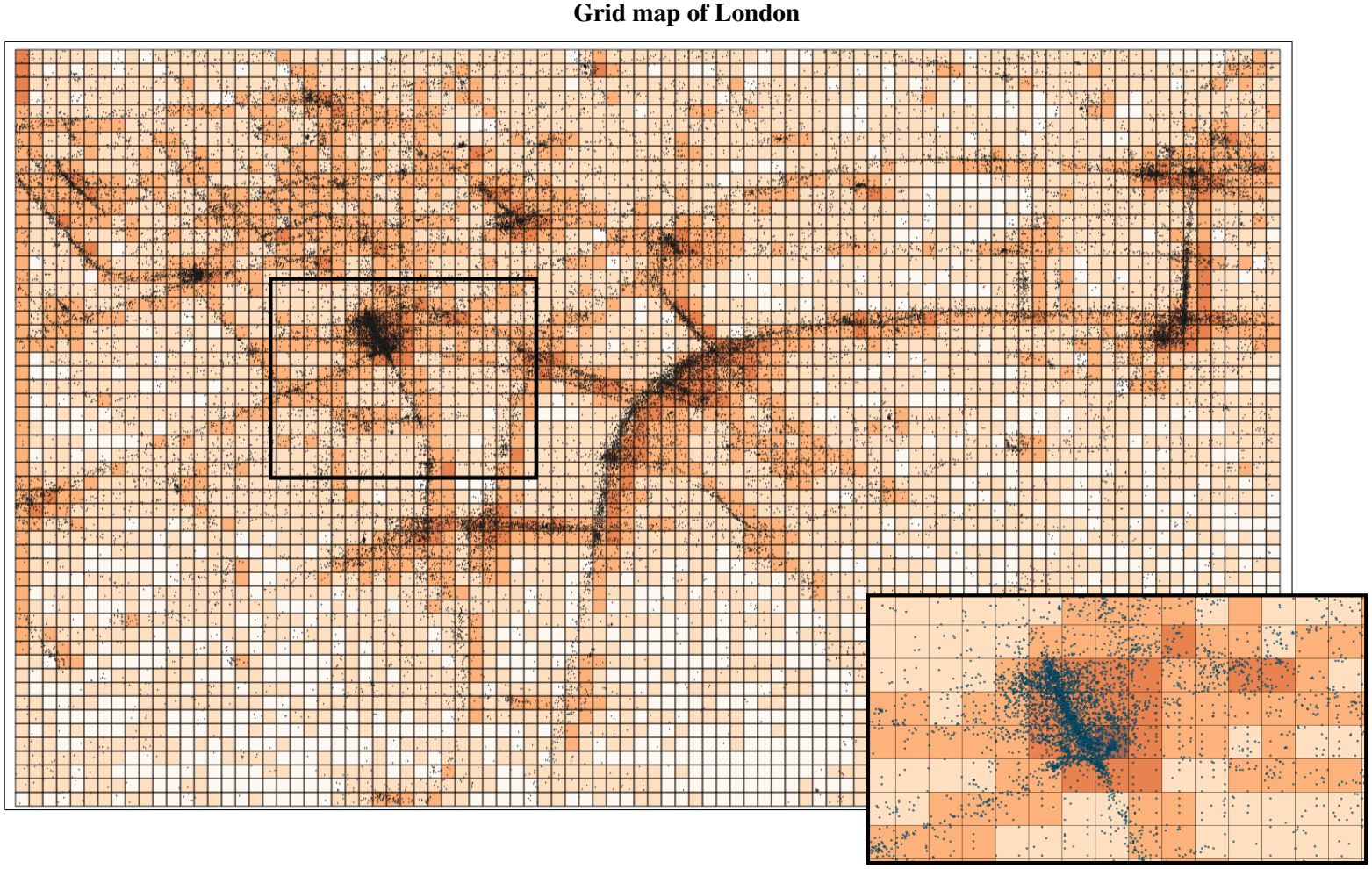


Fig. 12.17 A picture of London consisting of $20,130$ *squares* i.e. 183×110 in *width* $\times$ *height*. The picture is generated by dividing the bounding rectangle of the *large dataset* into smaller squares of equal area $2,500m^2$. Each square is coloured according to the number of points it contains; from white (none) to dark orange (densest).

12.6.5 Cluster density

In the section below, we present the cluster density $R(P)$ results of the algorithms. Note, we discarded all polygons with areas less than $1m^2$.

Table 12.4 presents the *averaged* areas, number of dots and density R for the algorithms. Overall, *for both datasets*, DBSCAN had the highest average density score, followed by k-MXT and mean shift; albeit the small differences. For mean shift, the average scores seem to be unaffected by the change in distance, which agrees with previous observation (see Figure 12.8). For k-MXT, the average area and number of dots increases with k while the average density R decreases. For DBSCAN, as $minPts$ decreases so does the average area, as expected. The average number of dots, on the other hand, does not always increase with area, which is probably due to the algorithm's noise filtering feature.

For the large dataset, Table 12.5 shows the polygons with the highest density according to each bucket size and parameter. As pointed out, the small polygons are less interesting and rather problematic. Hence, we bucket the polygons according to its areas as follows.

- *Small*: area between 100-1000 m^2;
- *Medium*: area between 1000-5000 m^2;
- *Large*: area above 5000 m^2.

Furthermore, we group the algorithms for comparison as shown previously in Table 12.1. We note that these groupings of k-MXT and DBSCAN($minPts$) are made subjectively, based on results from previous experiments. We group the pair of algorithms that produce results that are visually comparable i.e. same groups of points, similar polygons, see e.g Figure 12.18. In particular:

$$\begin{aligned} 1-\text{MXT} \quad & DBSCAN(minPts = 80pts); \\ 2-\text{MXT} \quad & DBSCAN(minPts = 40pts); \\ 3-\text{MXT} \quad & DBSCAN(minPts = 20pts); \\ 4-\text{MXT} \quad & DBSCAN(minPts = 3pts). \end{aligned}$$

Thus, the first column of Table 12.5 compares 1-MXT with DBSCAN($80pts$) and so on. For each *distance value* (10m, 25m, 50m), top-down cells correspond to densest polygons of each size, generated by the *same algorithm*. Looking by the columns from left to right i.e. 1-MXT to 4-MXT, show how the best scores and corresponding clusters change according to k and $minPts$. Furthermore, to examine the *accuracy* of these densest polygons, some are selected and plotted in Figure 12.18.

Overall, it can be seen that DBSCAN has the higher score with a minor exception at *25m - medium polygons*, in which k-MXT surpasses DBSCAN by a small margin. Looking from *left to right* i.e. 1-MXT/80pts to 4-MXT/3pts and also *top down* i.e. from small to large, it is seen that, generally, the

highest density reduces as polygon increases in size. This makes intuitive sense because typically the points/dots are more dense around specific locations (popular landmarks/attractions). Thus as the polygon gets larger, its density is reduced.

Looking at k-MXT at $10m$, *small* and *medium*, we see recurring polygons i.e. the same polygon [1] from Table 12.18 is produced with different values of k, a visualisation of such polygon can be found in Figure 12.18. An explanation is that these clusters achieved its maximum size for smaller values of k, thus unaffected when k increases. For example, it can be seen from Figure 12.8-(d) and Figure 12.11 in which there are some complete-overlapping polygons visible by its colours i.e. red and black boundaries overlapped. This can be interpreted as a feature of k-MXT.

We note that in most cases both algorithm identified the same groups of points.

	k-MXT				DBSCAN				MS			
		Avg. Area	Avg. Dots	Avg. R		Avg. Area	Avg. Dots	Avg. R	Avg. Area	Avg. Dots	Avg. R	
10m	1MXT	76	8.1	0.23	80pts	~~NA~~	~~NA~~	~~NA~~	9,052	115.1	0.017	**Avg. Small Set**
	2MXT	1,000	40.8	0.058	40pts	622	118.3	0.197				
	3MXT	2,490	77.7	0.043	20pts	1,111	109.6	0.108				
	4MXT	4,956	136.1	0.034	3pts	1,085	29.5	0.36				
25m	1MXT	303	12.3	0.148	80pts	5,141	285.6	0.054	9,305	123.5	0.014	
	2MXT	22,393	322.4	0.013	40pts	12,437	363.8	0.032				
	3MXT	90,299	1,053	0.011	20pts	29,650	469.3	0.021				
	4MXT	154,482	1,594.5	0.011	3pts	21,546	194.7	0.05				
50m	1MXT	610	14.6	0.104	80pts	53,517	946.6	0.017	13,667	167.6	0.013	
	2MXT	102,889	1,010.3	0.01	40pts	343,832	3,132	0.009				
	3MXT	166,928	1,608	0.01	20pts	184,903	1,629	0.006				
	4MXT	361,402	3,353	0.009	3pts	361,402	3,353	0.009				

	kMXT				DBSCAN				
		Avg. Area	Avg. Dots	Avg. R		Avg. Area	Avg. Dots	Avg. R	
10m	1MXT	63	6.9	0.514	80pts	1,467	440	0.336	
	2MXT	507	21.9	0.09	40pts	1,289	214	0.203	
	3MXT	1,459	40.8	0.06	20pts	1,127	117.7	0.123	
	4MXT	2,558	50.2	0.058	3pts	4,146	90.7	0.51	
25m	1MXT	327	10.3	0.159	80pts	9,087	523.6	0.056	**Avg. Large Set**
	2MXT	12,064	137.4	0.016	40pts	11,932	292.8	0.033	
	3MXT	101,353	686.3	0.008	20pts	19,306	235	0.021	
	4MXT	248,969	1,441.5	0.007	3pts	107,536	594.2	0.032	
50m	1MXT	745	13.2	0.11	80pts	77,196	948.2	0.014	
	2MXT	112,369	662.8	0.006	40pts	168,737	1,053	0.008	
	3MXT	1,982,196	10,417.7	0.004	20pts	182,290	1,057.3	0.006	
	4MXT	4,387,805	21,603	0.004	3pts	144,211	713.2	0.02	

Table 12.4 The above tables show the *average score* of R for each corresponding algorithm. **Top table** shows results on the *small dataset*. **Table below** shows results on the *large dataset*. The reference values are $R(\text{small}) = 0.005$ and $R(\text{large}) = 0.003$

	Polygon	Algorithm	1-MXT/DBSCAN(80pts)				2-MXT/DBSCAN(40pts)				3-MXT/DBSCAN(20pts)				4-MXT/DBSCAN(3pts)			
			Area	Num Dots	R	Label	Area	Dots	R	Label	Area	Dots	R	Label	Area	Dots	R	Label
10m	Small	kMXT	161	64	0.397		120	35	0.291		120	35	0.291		120	35	0.291	
		DBSCAN	206	85	0.412		106	48	0.45		409	99	0.242		105	22	0.208	
	Medium	kMXT	~~NA~~	~~NA~~	~~NA~~		2034	167	0.082	**[1]**	2034	173	0.085	**[1]**	2034	173	0.085	**[1]**
		DBSCAN	4,673	1,359	0.291	**[g]**	1,426	273	0.191	**[e]**	4,233	495	0.117	**[c]**	1,374	75	0.055	**[a]**
	Large	kMXT	~~NA~~	~~NA~~	~~NA~~		32,944	2,450	0.074	**[2]**	61,993	3,401	0.055	**[3]**	126,971	3,914	0.031	**[4]**
		DBSCAN	~~NA~~	~~NA~~	~~NA~~		14,506	2,155	0.149	**[f]**	20,001	2,730	0.136	**[d]**	64,666	3,473	0.054	**[b]**
25m	Small	kMXT	134	45	0.335		140	23	0.163		~~NA~~	~~NA~~	~~NA~~		~~NA~~	~~NA~~	~~NA~~	
		DBSCAN	~~NA~~	~~NA~~	~~NA~~		807	42	0.052		261	26	0.1		104	5	0.048	
	Medium	kMXT	1,123	64	0.057	**[5]**	1,690	84	0.05	**[6]**	1,049	52	0.05	**[8]**	1,782	40	0.022	**[10]**
		DBSCAN	8,965	607	0.068	**[o]**	2,566	114	0.04	**[m]**	2,671	115	0.043	**[k]**	8,520	130	0.015	**[h]**
	Large	kMXT	~~NA~~	~~NA~~	~~NA~~		17,237	682	0.04	**[7]**	13,307	204	0.015	**[9]**	13,090	124	0.009	**[11]**
		DBSCAN	30,080	3,052	0.101	**[p]**	45,710	3,310	0.072	**[n]**	41,127	1,093	0.027	**[ℓ]**	11,432	122	0.011	**[i]**
50m	Small	kMXT	111	54	0.486		~~NA~~	~~NA~~	~~NA~~		~~NA~~	~~NA~~	~~NA~~		~~NA~~	~~NA~~	~~NA~~	
		DBSCAN	~~NA~~	~~NA~~	~~NA~~		~~NA~~	~~NA~~	~~NA~~		~~NA~~	~~NA~~	~~NA~~		112	5	0.045	
	Medium	kMXT	2,460	198	0.08	**[12]**	3,875	121	0.031	**[14]**	~~NA~~	~~NA~~	~~NA~~		~~NA~~	~~NA~~	~~NA~~	
		DBSCAN	6,993	101	0.014		8,714	103	0.012		2,568	29	0.011		1,664	16	0.01	
	Large	kMXT	16,376	27	0.002	**[13]**	22,317	340	0.015	**[15]**	6,881,951	38,019	0.006	**[16]**	8,395,736	42,616	0.005	
		DBSCAN	178,185	4,377	0.025		31,686	381	0.012		15,570	136	0.009		8,394,908	43,084	0.005	

Table 12.5 The above table shows the **highest R score** for each corresponding type of polygon: *small* with area between 100-1000 m^2; *medium*: 1000-5000 m^2 and *large* above 5000 m^2. ~~NA~~ indicates no observed data exists for that parameter.
Some polygons are *labelled* using *letters* for DBSCAN, and *numeric* for *k*-MXT. These polygons are plotted in Figure 12.18 with its associated labels for identification. The remaining polygons are excluded because they either overlap with those selected or difficult to visualise (very small).

Highest ranked polygons from Table 12.5.

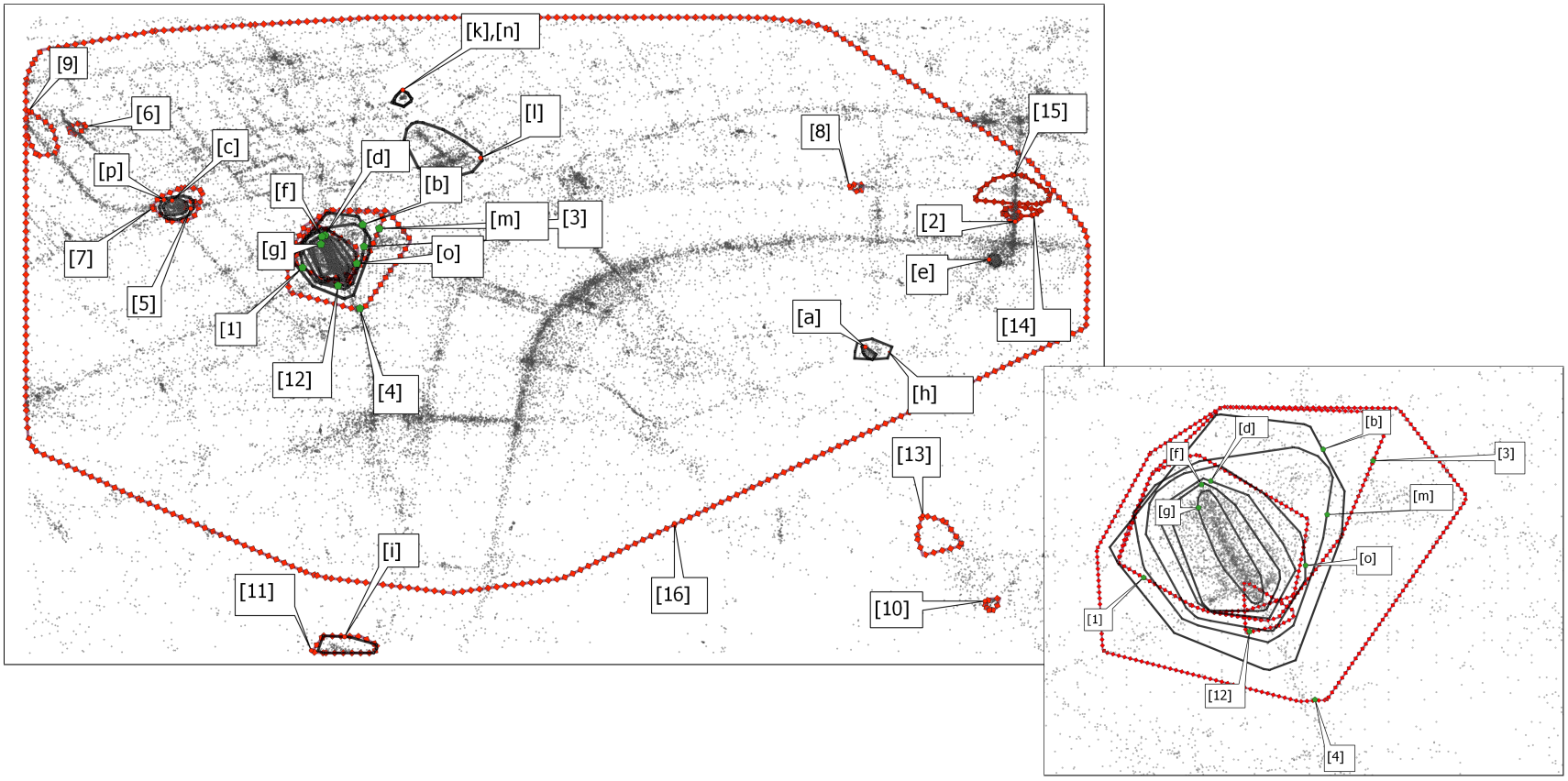


Fig. 12.18 The figures above show the highest ranked polygons presented in Table 12.5. Each polygon has a unique label for identification (see Table 12.5). Furthermore, each polygon is drawn by: *black, solid line* for DBSCAN, and *red, marked (diamond shape) line* for k-MXT. Additionally, the small figure (right hand side) provides a closer look at the Trafalgar Square region, which *attracts* most of the polygons.

12.7 Parameterised k-MXT

As discussed in previous section, for higher values of k, the density of clusters produced by k-MXT is significantly reduced. This is because data points are not evenly distributed as they are biased toward specific locations (landmarks, attractions, etc.). Thus the expansion of the polygons (equivalently k) inevitably reduces the density. Furthermore, this behaviour is due to the absence of a *noise controlling mechanism*. The small polygons are considered as small clusters so far, not noise. To improve the k-MXT algorithm, we introduce a parameter w.

The parameterised algorithm k-MXT(w_{min}) adds a simple *filtering* in the edge selection process i.e. for an edge e with weight $w(e)$ then

if $w(e) < w_{min}$ **then**
  ignore ;
else
  consider ;
end

Thus, every edge e' that $w(e') < w_{min}$ is considered a *noise* edge and excluded. This would reduce the size of the polygons by removing points located towards the boundary, whence the density is significantly sparser.

12.7.1 Results

We perform clustering with the parameterised k-MXT on the large dataset. We use 2-MXT($d = 25m$) which previously showed some promising results however required improvement (see Figure 12.11). Table 12.6 presents the average density and the density of the top polygons for each size as in previous sections. In comparison, results from the k-MXT algorithm i.e. 2-MXT($w = 0$) and DBSCAN($minPts = 40, 80$) are also included.

Additionally, Figure 12.19 plots selected clusters over the region of Trafalgar Square. These clusters are produced by k-MXT($w = 40, 80$) to compare directly with DBSCAN($minPts = 40, 80$), with $d, \varepsilon = 25m$. Overall, it can be seen that the introduction of w significantly improves the results of k-MXT.

	Density	2-MXT	2-MXT ($w = 40$)	DBSCAN ($minPts = 40$)	2-MXT ($w = 80$)	DBSCAN ($minPts = 80$)
Small Area: $[100, 1000]\ m^2$	Best Average	0.163 0.055	0.215 0.08	0.052 0.045	0.274 0.12	NA NA
Medium Area: $[1000, 5000]\ m^2$	Best Average	0.05 0.013	0.103 0.052	0.044 0.031	0.113 0.086	0.068 0.054
Large Area: $> 5000\ m^2$	Best Average	0.04 0.01	0.101 0.068	0.072 0.034	0.128 0.128	0.101 0.062

Table 12.6 Average density on the large dataset with $d, \varepsilon = 25m$.

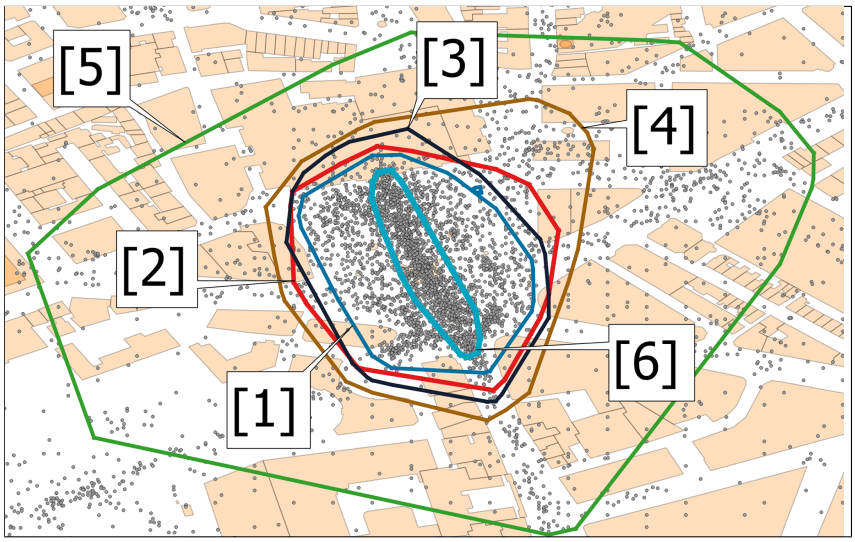
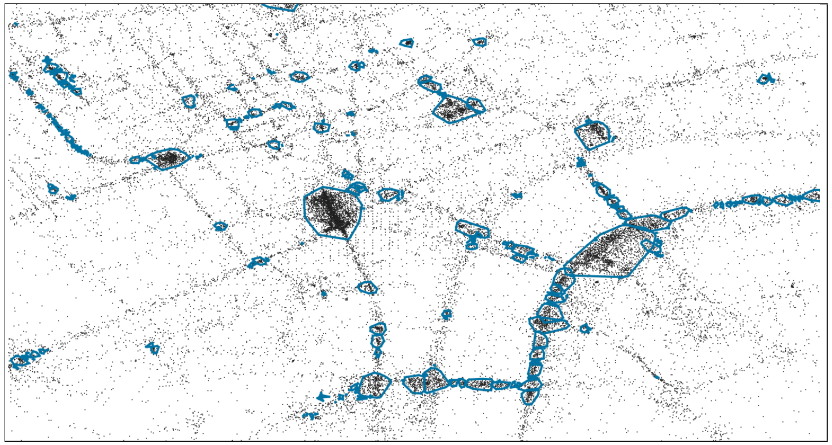


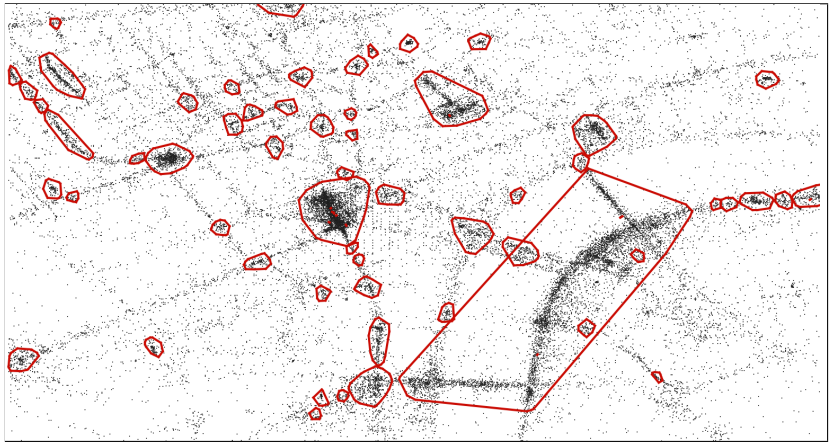
Fig. 12.19 Region of Trafalgar Square. The parameters and resulting density R for each polygon are:

- [1]-blue 2-MXT($d = 25m, w = 80$), $R = 0.128$;
- [2]-red 2-MXT($d = 25m, w = 40$), $R = 0.101$;
- [3]-black DBSCAN($\varepsilon = 25m, minPts = 80$), $R = 0.101$;
- [4]-brown DBSCAN($\varepsilon = 25m, minPts = 40$), $R = 0.072$;
- [5]-green 2-MXT($d = 25m, w = 0$), $R = 0.026$;
- [6]-turqoise DBSCAN($\varepsilon = 10m, minPts = 80$), $R = 0.291$.

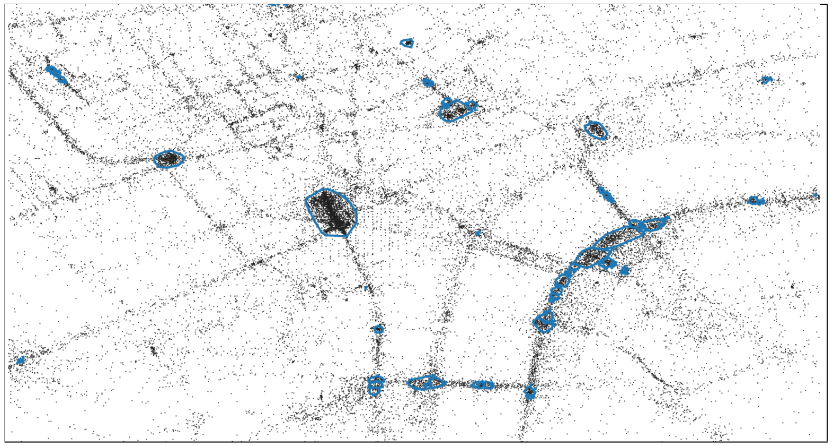
k-MXT(w) and DBSCAN on large dataset with $d, \varepsilon = 25m$



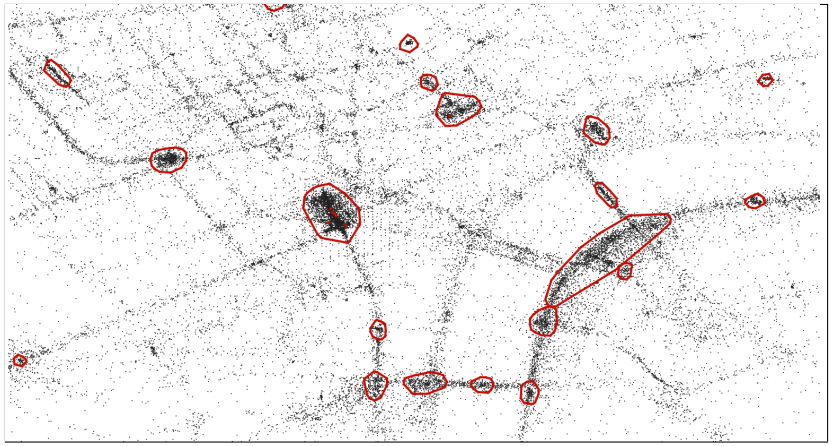
(a) 2-MXT($w = 40$)



(b) DBSCAN($minPts = 40$)



(c) 2-MXT($w = 80$)



(d) DBSCAN($minPts = 80$)

Fig. 12.20

12.8 Blind test

We conclude this chapter by presenting the results from our *blind tests*. In which, we randomly select a city then cluster it and examine the results. We found that some polygons appear to contain popular tourist attractions in these cities, which are then labelled accordingly.

Blind test [1]

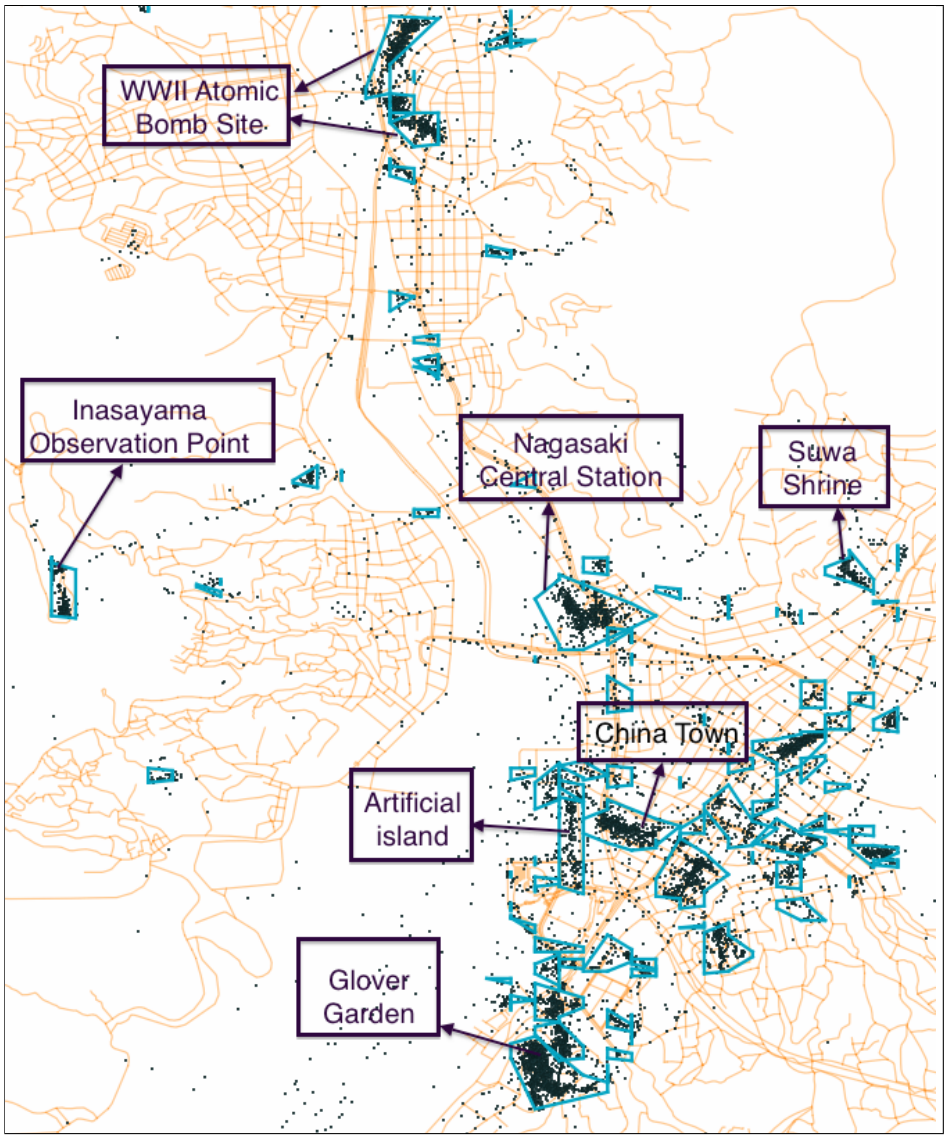


Fig. 12.21 The city in question is Nagasaki, Japan. The algorithm applied is 2-MXT($d = 25m, w = 0$). The dense regions covered in the polygons appear to be associated with some popular tourist attractions e.g. WWII Atomic Bomb site, China Town, Glover Garden, Suwa Shrine, etc.

Blind test [2]

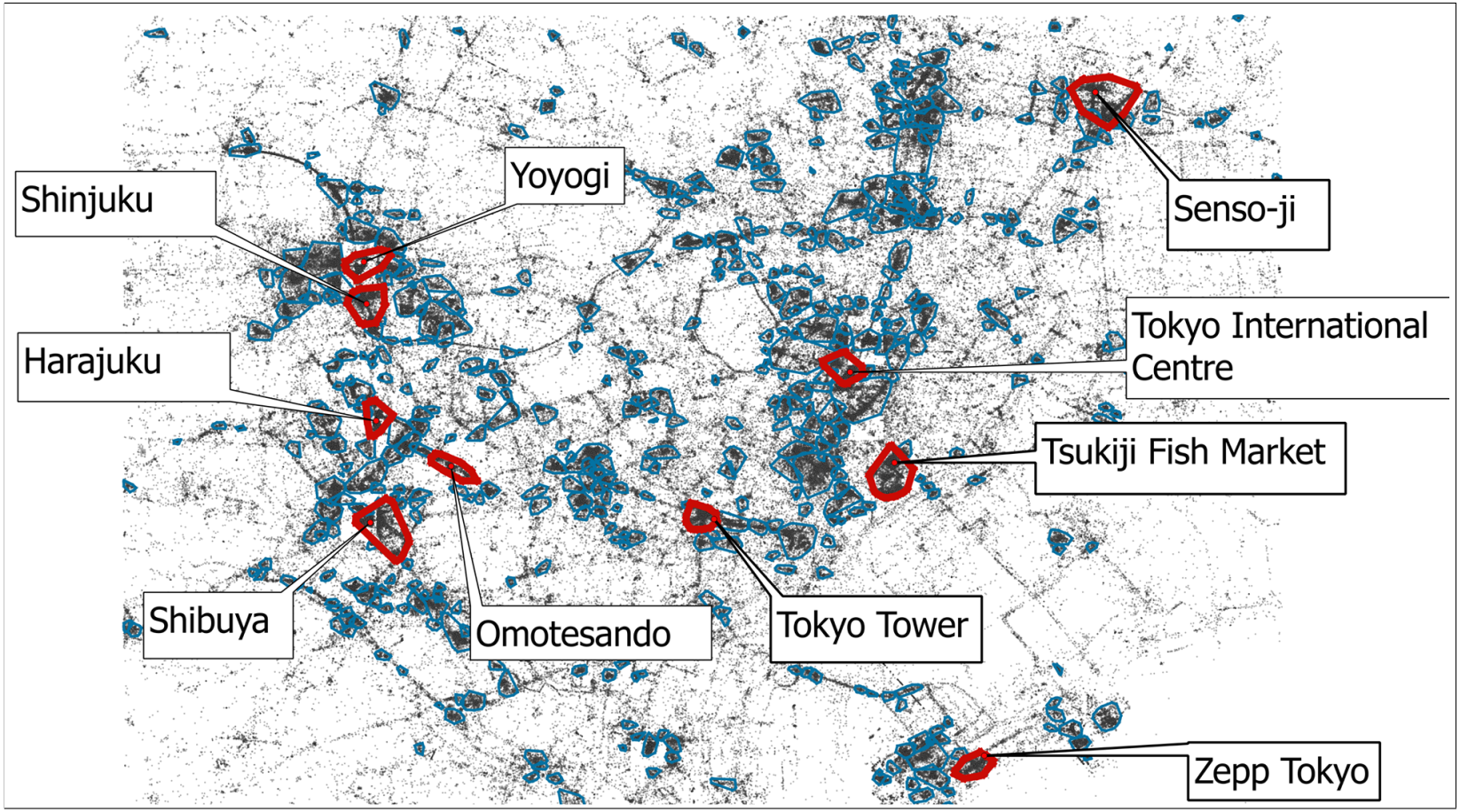


Fig. 12.22 The city in question is Tokyo, Japan. The algorithm used is 2-MXT($d = 50m, w = 20$). The highlighted polygons are *ten* polygons with the highest density. We then attempt to identify the landmarks/attractions associated with these clusters. It turns out that some regions contained inside the polygons are popular points of interest in Tokyo e.g. Tokyo tower, Shibuya crossing, Shinjuku, Senso-ji temple, etc.
In this experiment, the distance $50m$ appears to produce visually better results. This is because, as the area of central Tokyo is much larger compares to London's, we are observing at a scale much higher than in previous figures. And therefore, larger polygons are more noticeable.

Summary

This research began from ideas that sparked in experiments with *connecting the dots* pictures. Using pictures made of purely *dots*, graphs can be constructed by simply connecting a pair of dots which are located in proximity of each other's. For *dots* which are coordinates of photographs, the resulting graphs appear to have community structures. That is, the subgraphs at some locations are densely connected; also these subgraphs might be connected however by much sparser subgraphs. This phenomenon reflects the fact that the coordinates of photographs are distributed with some unknown underlying probability distribution but nevertheless bias towards *popular photo-locations*. Thus, it is natural to follow up with the question: how can we break the graph into *fragments* in such a way that its cluster structure is preserved?

Consider a fun thought experiment as follows. Imagine the vertices as particles and suppose there is a force connecting each pair of connected particles. Further the strength of the force increases as a function of the number of connected particles in the group. Then for a group made of many particles, its formation is so robust that it can withstand any interruptive force and maintain its structure. In which case, if we *smash* the *graph* using, say a hammer, with a sufficient force to break the weak connections between the groups of particles. We would then be able to collect these separated, tightly-connected groups of particles or *fragments*. This thought experiment provides the basis for the main topic of this research, for which we call *graph fragmentation* algorithms.

Graph fragmentation is a procedure for decomposing an input graph into connected subgraphs or *fragments*. The basic decomposition procedure is simple: a vertex *attracts* and *absorbs* a neighbour, the process is carried out until there remains only vertices with no neighbour. This process can be easily modified by changing the decision on e.g. how vertices are selected, which vertex absorbs the other, and so on. We studied and analysed a number of fragmentation variations in Part I and Part II. For most of these variations, the *conditional absorption rules* proved to be problematic as it is difficult to obtain analytical results for these models even on simple graph model i.e. cycle. In this sense, the *permutation subgraph* seems more robust as we were able to obtain some properties of these permutation fragments on a variety of graph models such as cycle, r-regular graph.

We dedicated some attention on analysing the permutation subgraph of random graphs and were able to obtain some basic yet interesting properties of the model. Notably when apply on the extended infinite random graph model, the limiting distribution of the number of left children of vertices is shown to be Poisson with parameter $\lambda = 1$. This property in addition to the process of generating the

graph itself together show a slight similarity (hence perhaps hint at some relationship, however not yet proven) between the permutation subgraph and birth–death/branching process [63] or conditioned Galton-Watson forest [29]. Thus, a possible direction of research for the *permutation subgraph* is, firstly, to extend its results on a finite random graph for instance an interesting property would be the distribution of the component sizes. Secondly is to obtain further results on infinite random graphs, the result obtained here only applies for a small range of value of the number of children k. It would be of interest to look at a wider range for values of k, and if possible to draw some comparison with the branching process, a seemingly connected model would be the aforementioned conditioned Galton-Watson forest considered by [29].

In Part III, we studied the k-MXT algorithm (a fragmentation variation) and evaluated it as a graph clustering tool. Analytical results on the planted ℓ partitions model show that the algorithm works well for $k \geq 2$ on graphs with sufficient number of triangles. However, empirical results show that on real-networks or sparser computer generated networks the algorithm is generally out-performed by some of current best algorithms. This is because, in most of these graphs, the triangle density in these graphs is often less than the required threshold in order for k-MXT to perform well. Further, by its nature the algorithm can only produce *hard-clustering* which is rather rigid. These points indicate the limitations of the k-MXT algorithm as a community detection algorithm. Nevertheless, some empirical evidences show that the parameterised version k-MXT(w) produces clustering in which subgraphs seem to possess properties of a small-world network e.g. high clustering coefficient, low diameter, etc. Thus, this can be a potential direction of research for k-MXT as a graph clustering algorithm.

The final chapter is devoted to evaluate the k-MXT algorithm as a geometric clustering algorithm, specifically on datasets generated from photograph's coordinates - the original motivation of the research. Using visual inspection and a simple measure, it is shown that the results of k-MXT(w) is comparable to that of DBSCAN, one of the most popular clustering algorithms. This is due to the fact that the geometric/proximity graphs generated in this setting contains a *great number of triangles*. Thus, applying k-MXT(w) with increasing w allow us to *filter out* the out-lying dots and concentrating the regions of dots into several dense clusters. Since k-MXT(w) works on the basis of graphs, it is evident that graph modelling is applicable in this setting and further it is presumable that other graph clusterings would also be able to achieve significant results. This is perhaps due to an underlying close relationship between *triangles, graphs* and *geometric* see for example Section 3.6 in [54]. Thus, for future work, an analytical study of fragmentation algorithms on *geometric* graphs would be greatly beneficial. Essentially, the *dots* are two-dimensional data, hence *fragmentation* and *community detection* algorithms might also be applicable for clustering *high-dimensional* data which requires further investigations. A foreseen disadvantage of graph modelling approach to this type of problem is the complexity for constructing the *graph*, which can also be an interesting and practical problem for future research.

Appendix A

Generating functions

In this section, we provide some material about the theory of *generating functions* which are used in the main content of the study, especially in Chapter 3 and Chapter 6. Most of the material provided below follows the book *Generatingfunctionology* due to Wilf [63].

'A generating function is a clothesline on which we hang up a sequence of numbers for display', Wilf.

A.1 Generating function

Suppose we are given a sequence of numbers $a_0, a_1, a_2, a_3, \ldots$, and we want to know what this sequence is? To answer this, we are often concerned with the problem of finding a closed form for the n^{th} term a_n. If the sequence is easy to guess for instance $2, 4, 6, 8, 10, 12\ldots$ then it is a sequence of even numbers and that the n^{th} term is $a_n = 2n$. However, there are many cases where it might not be possible to obtain a closed form for the a_n member of the sequence, for instance consider the sequence of *prime numbers*.

In some cases, although obtaining a simple formula for the members of the sequence may be problematic, it might be possible to obtain a formula for *the sum of a power series*, whose i^{th} coefficient is the i^{th} number in the sequence that we are examining. A power series is an expression of the form

$$\sum_{n=0}^{\infty} a_n x^n = a_0 + a_1 x^1 + a_2 x^2 + \ldots$$

Consider the sequence $0, 1, 1, 2, 3, 5, 8, 13, \ldots$, this is the well-known Fibonacci numbers that satisfy the recurrence

$$F_n = F_{n-1} + F_{n-2} \qquad (F_0 = 1, F_1 = 1, n \geq 1)$$

Suppose we have a power series in which the coefficient of the n^{th} power x^n is the n^{th} Fibonacci number F_n. Then the sum of this infinite series is the generating function $G(x)$ of the Fibonacci

sequence

$$G(x) = \sum_{n=0}^{\infty} F^n x^n = F_0 + F_1 x^1 + F_2 x^2 + F_3 x^3 + \dots \qquad (F_0 = 1, F_1 = 1, n \geq 1)$$

It turns out we can obtain a closed form for the generating function of $G(x) = x/(1-x-x^2)$, which gives an approximate formula for the n^{th} coefficient i.e. the n^{th} Fibonacci number: $F_n \sim \frac{1}{\sqrt{5}}\left(\frac{1+\sqrt{5}}{2}\right)^n$ [63]. The exact value for the 30^{th} Fibonacci number is 832040, while a numerical approximation using the formula is 832040.00000024.

In the majority of Chapter 3, we try to solve a *recurrence relation* using generating functions. Typically, we begin by introducing a generating function. We then go through the many manipulations steps that include finding the derivatives of that generating function. The problem is that it is not certain whether the various series will converge to a function. Nevertheless, Wilf shows that the various manipulations of generating functions can be carried out in the ring of formal power series, where questions of convergence are nonexistent [63]. Thus we can execute the whole method and end up with the generating series. If the series converges, we can perhaps obtain a nice *answer* to the problem. However, if not, we might not be able to get analytic information, such as asymptotic formulas for the sizes of the coefficients. Which turns out to be crucial for getting the results throughout Chapter 3.

A.1.1 Generating function manipulations

There are many types of generating functions. In the majority of this study, we mainly deal with *ordinary generating functions* (OGFs) (in Chapter 3 and Chapter 6). We also encounter the *exponential generating function* EGF, however very briefly in Chapter 5. Thus, the operations we list below mainly apply to OGF.

Recall the definition: given the sequence $a_0, a_1, a_2, \dots$, the function

$$G(x) = \sum_{n=0}^{\infty} a_n x^n$$

is an OGF of the sequence, we say $G(x)$ *generates* the sequence $\{a_n\}_0^{\infty}$. We use the notation $[x^n]G(x)$ to denote the n^{th} coefficient of the series a_n. The first simple manipulation of OGF is addition or subtraction, let there be another OGF $F(x) = \sum_n b_n x^n$, then

$$G(x) \pm F(x) = \sum_n a_n x^n \pm \sum_n b_n x^n = \sum_n (a_n \pm b_n) x^n$$

A.1.1.1 Left shifting

Suppose $G(x)$ generates $\{a_n\}_0^{\infty}$, then what series generates $\{a_{n+1}\}_0^{\infty}$. Let this be $F(x)$, we have

$$F(x) = \sum_{n=0}^{\infty} a_{n+1} x^n = a_1 + a_2 x + a_3 x^2 + \dots = \frac{1}{x}(a_1 x + a_2 x^2 + a_3 x^3 + \dots) = \frac{G(x) - a_0}{x}$$

Thus by shifting the series by 1 unit to the left, we obtain a generating function for the series $\{a_{n+1}\}_0^\infty$. It follows that

$$F'(x) = \Big(\frac{G(x)-a_0}{x} - a_1\Big)/x = \frac{G(x)-a_0-a_1x}{x^2} = a_2x + a_3x^2 + a_4x^2 + \cdots = \sum_{n=0}^{\infty} a_{n+2}x^n$$

and we obtain a series generates the sequence $\{a_{n+2}\}_0^\infty$.

Thus, generally, given an OGF $G(x)$ generates $\{a_n\}_0^\infty$, we can obtain a series $F(x)$ for $\{a_{n+h}\}_0^\infty$ by left-shifting $G(x)$ k times

$$F(x) = \frac{G(x)-a_0-a_1x-\cdots-a_{h-1}x^{h-1}}{x^h}$$

A.1.1.2 Right shifting

Consider

$$xG(x) = x\sum_{n=0}^{\infty} a_nx^n = \sum_{n=1}^{\infty} a_{n-1}x^n$$

thus

$$x^hG(x) = x^h\sum_{n=0}^{\infty} a_nx^n = \sum_{n=h}^{\infty} a_{n-h}x^n$$

A.1.1.3 Index multiply

Suppose $G(x)$ generates $\{a_n\}_0^\infty$, an important question is what generates $\{na_n\}_0^\infty$. First, consider the derivative

$$G'(x) = a_1 + 2a_2x + 3a_3x^2 + \cdots = \sum_{n=0}^{\infty}(n+1)a_{n+1}x^n$$

The difference between the coefficient and its power is 1, thus with a right shift

$$xG'(x) = a_1x + 2a_2x^2 + 3a_3x^3 + \cdots = \sum_{n=1}^{\infty} na_nx^n$$

A.1.1.4 Convolution

Let there be $G(x)$ generates $\{a_n\}_0^\infty$ and $F(x)$ generates $\{b_n\}_0^\infty$, the convolution of the two series is

$$\begin{aligned}
G(x)F(x) &= \sum_n a_nx^n \sum_n b_nx^n = \Big(a_0 + a_1x + a_2x^2 + \ldots\Big)\Big(b_0 + b_1x + b_2x^2 + \ldots\Big) \\
&= a_0\Big(b_0 + b_1x + b_2x^2 + \ldots\Big) + a_1x\Big(b_0 + b_1x + b_2x^2 + \ldots\Big) + a_2x^2\Big(b_0 + b_1x + b_2x^2 + \ldots\Big) + \ldots \\
&= a_0b_0 + (a_0b_1 + a_1b_0)x + (a_0b_2 + a_1b_1 + a_2b_0)x^2 + \ldots \\
&= \sum_{n=0}^{\infty}\Big(\sum_{k=0}^{n} a_kb_{n-k}\Big)x^n
\end{aligned}$$

A.1.1.5 Partial Sum

Let there be $G(x)$ that generates the sequence $\{a_n\}_0^\infty$. Suppose $F(x)$ is the geometric series i.e. $F(x) = \sum_{n=0}^{\infty} b_n x^n$ where $b_n = 1$ for $n \geq 0$. Thus by the previous rule of convolution, we have

$$G(x)F(x) = \sum_{n=0}^{\infty} \Big(\sum_{k=0}^{n} a_k \Big) x^n$$

Further, since $F(x) = 1/(1-x)$ hence the effect of dividing an OGF by a factor of $(1-x)$ is to create a new sequence in which the coefficient is the its partial sums.

A.2 Asymptotic of the coefficient of a power series

As pointed out that power series can be manipulated without necessarily worrying about whether the resulting series converge. Nevertheless, if the series converge and they represent functions, then we might be able to find the analytic properties of the series and their coefficient sequences. This turns out to be useful in our study, since the series we that derived through various manipulations do not have closed form and perhaps problematic to expand. Thus, we often rely on the following theorems about the asymptotic of the coefficient to obtain our results.

A.2.1 Theorems on analyticity and asymptotic of generating functions

Below we present the important theorems regarding the analyticity and asymptotic of generating functions. Since the proofs are involved, they are omitted, readers can visit the source material for more details.

Let there be a power series $f = \sum_{n=0}^{\infty} a_n z^n$ where z is a complex variable. Then

Theorem 10. *[Theorem 2.4.1 [63]] There exists a number R, $0 \leq R \leq +\infty$, called the radius of convergence of the series f, such that the series converges for all values of z with $|z| < R$ and diverges for all z such that $|z| > R$. The number R is expressed in terms of the sequence $\{a_n\}_0^\infty$ of coefficients of the series by means of*

$$R = \frac{1}{\limsup_{n\to\infty} |a_n|^{1/n}} \qquad (1/0 = \infty; 1/\infty = 0)$$

The *limit superior* (lim sup) is defined as follows. Let there be a sequence $\{a_n\}_0^\infty$ then L is the limit superior of the sequence if

1. L is finite and
   (a) for any $\varepsilon > 0$ there is a *finite* number of terms a_n such that $a_n < L + \varepsilon$,
   (b) for any $\varepsilon > 0$ there is an *infinite* number of terms a_n such that $a_n > L - \varepsilon$, or
2. $L = +\infty$ and for any $x > 0$ there is a term $a_n > x$, or

3. $L = -\infty$ and for any y there is a *finite* number of terms such that $a_n > y$.

Theorem 11. *[Theorem 2.4.2 [63]] Suppose the power series f converges for all z in $|z| < R$, and let $f(z)$ denote its sum. Then $f(z)$ is an analytic function in $|z| < R$. If furthermore the series diverges for $|z| > R$, then the function $f(z)$ must have at least one singularity on the circle of convergence $|z| = R$.*

Consider the geometric series: $f(z) = 1 + z + z^2 + \cdots = \sum_{n=0}^{\infty} z^n$ (hence $a_n = 1$ for every $n \geq 0$), it is known that the series converges for $|z| < 1$ and diverges for $|z| > 1$. Thus it must be that the series has a radius of convergence $R = 1$. This is true since $f(z) = \sum_{n=0}^{\infty} z^n = 1/(1-z)$, hence the series has a singularity at $z = 1$.

Theorem 12. *[The asymptotic of the coefficients of a power series] Let $f(z)$ be analytic in some region containing the origin, let a singularity of $f(z)$ of smallest modulus be at a point $z_0 \neq 0$, and let $\varepsilon > 0$ be given. Then there exists N such that for all $n > N$ we have*

$$|a_n| < \left(\frac{1}{|z_0|} + \varepsilon\right)^n \tag{A.1}$$

Further, for infinitely many n we have

$$|a_n| > \left(\frac{1}{z_0} - \varepsilon\right)^n$$

In this study, especially in Chapter 3 and Chapter 6, we usually obtain a function $f(z)$ that generates a sequence of numbers of interest i.e. the expected number of components of a graph. The problem is that it is problematic to expand the series to obtain a formula for the coefficients of the series. Thus, alternatively, we want to find the asymptotic behaviour of the sequence.

It is often the case that the function that we can obtain $f(z)$ has a single singularity which is a pole at $z = z_0$ (i.e. $f(z) \to \infty$ as $z \to z_0$). By Theorem 11, z_0 is the radius of the convergence of $f(z)$, and $f(z)$ is analytic in the disk $|z| < z_0$ in which its power series expansion about the origin converges. Theorem 12 then states that, for the n^{th} coefficient of the series

$$|a_n| < \left(\frac{1}{|z_0|} + \varepsilon\right)^n$$

However, this might offer little information about the growth of the coefficients. In which case, the strategy is to find another function $g(z)$ which has the same singularity at z_0. Then by subtracting $g(z)$ from $f(z)$, we obtain a function $f(z) - g(z)$ which is analytic in a larger disk with radius R'. Hence the power series coefficients of the new function is $< \left(\frac{1}{R'} + \varepsilon\right)^n$, which becomes smaller thus we might obtain some new insights about its growth.

Suppose the function $f(z)$ is meromorphic (it is analytic except for finite number of *poles*), let z_0 be a pole of $f(z)$ of order r, $1 \leq r < \infty$. Then in some punctured disk centred at z_0 $f(z)$ has an expansion

$$f(z) = \sum_{j=1}^{r} \frac{a_{-j}}{(z-z_0)^j} + \sum_{i=0}^{\infty} a_i (z-z_0)^i \tag{A.2}$$

The above series expansion is known as the *Laurent series* [24] and the first summand is the *principal parts* of the Laurent series about the singularity z_0. Thus, by subtracting off the principal parts, the function $g(z) = f(z) - \sum_{j=1}^{r} \frac{a_{-j}}{(z-z_0)^j}$ is analytic at z_0. Thus, we extend the radius of convergence from $|z_0| = R$ to a larger disk of radius $R' > R$.

In our studies, in most cases by subtracting off the principal parts, we obtain a new function $g(z)$ which is analytic in the whole plane i.e. its radius of convergence is $R' = \infty$. Thus, by Theorem 12 the power series coefficients of the function $g(z)$ about the origin is $< (\frac{1}{R'} + \varepsilon)^n = O(\varepsilon^n)$ for any $\varepsilon > 0$. We see that

$$[x^n]g(z) = [x^n]f(z) - [x^n] \sum_{j=1}^{r} \frac{a_{-j}}{(z-z_0)^j} < O(\varepsilon^n)$$

thus the growth of the coefficients of $f(z)$ is well approximated by the coefficients of its principal parts.

A.2.1.1 Example

Consider the function $f(z) = e^z/(z-1)$. The function e^z is an entire function, meaning its radius of convergence is infinite. However, by dividing it with $(z-1)$ the function has a pole at $z = 1$. Thus, to obtain its principal parts, we expand the series about the point $z_0 = 1$

$$\begin{aligned} f(z) &= e\frac{1}{(z-1)}e^{z-1} = e\frac{1}{(z-1)}\Big(1 + (z-1) + \frac{(z-1)^2}{2!} + \frac{(z-1)^3}{3!} + \dots\Big) \\ &= e\Big(\frac{1}{z-1} + 1 + \frac{(z-1)}{2!} + \frac{(z-1)^2}{3!} + \dots\Big) \end{aligned}$$

Thus the principal part of $f(z)$ is $g(z) = e/(z-1)$. Since, e is a constant and $1/(z-1)$ is the geometric series with a negative sign, hence $[x^n]g(z) = -e$. Denote by a_n the coefficients of $f(z)$ and by b_n the coefficients of $g(z)$. Let $h(z) = f(z) - g(z)$, then $h(z)$ is an entire function hence by Theorem 12 $|a_n - b_n| < O(\varepsilon^n)$ for $n \to \infty$. Thus, $a_n \sim b_n = -e$.

On the other hand, applying the *partial sum* rule (see Section A.1.1.5), we get

$$f(z) = \frac{e^z}{(z-1)} = -\sum_{n=0}^{\infty}\Big(\sum_{k=0}^{n}\frac{1}{k!}\Big)x^n \quad \text{hence} \quad a_n = -\sum_{k=0}^{n}\frac{1}{k!},$$

thus we have approximated the n^{th} partial sum of e with e itself.

A.2.1.2 An useful technique used in the study

Suppose now that we have the function $f(z) = \frac{\varphi(z)}{z-1}$ where $\varphi(z)$ is an entire function. Write

$$f(z) = \frac{\varphi(1)}{z-1} + \frac{\varphi(z) - \varphi(1)}{z-1}$$

By the L'Hospital rule we can get

$$\lim_{z\to 1}\frac{\varphi(z)-\varphi(1)}{z-1}=\frac{\varphi'(z)}{1}<\infty$$

as φ is entire, hence $\frac{\varphi(z)-\varphi(1)}{z-1}$ is analytic. So

$$[x^n]f(z)=[x^n]\frac{\varphi(1)}{z-1}+[x^n]\frac{\varphi(z)-\varphi(1)}{z-1}$$

as $n\to\infty$ hence $[x^n]f(z)\sim[x^n]\frac{\varphi(1)}{z-1}=-\varphi(1)$.

Applying the above technique to the previous example, we have

$$f(z)=\frac{e^z}{z-1}=\frac{e}{z-1}+\frac{e^z-e}{z-1}$$

Since the second function is entire, its radius of convergence is thus infinity, thus

$$[x^n]f(z)=-e+O(\varepsilon^n)\sim -e$$

as $n\to\infty$.

A.3 Some useful power series

A.3.1 Geometric series

$$\frac{1}{1-x}=\sum_{n\geq 0}x^n \tag{A.3}$$

A.3.2 Logarithm

$$\log\frac{1}{1-x}=\sum_{n\geq 1}\frac{x^n}{n} \tag{A.4}$$

A.3.3 Exponential

$$e^x=\sum_{n\geq 0}\frac{x^n}{n!} \tag{A.5}$$

A.3.4 Binomial expansion

$$(1+x)^\alpha=\sum_k\binom{\alpha}{k} \tag{A.6}$$

$$\frac{1}{(1-x)^{k+1}}=\sum_n\binom{n+k}{n}x^n \tag{A.7}$$

A.4 Procedure for solving linear first order differential equation

In many cases, the derivation of a generating function requires us to solve a *linear differential equations*. There is a formula for the general solution of a functional equation of such form [36] [61]. We discuss this below.

Suppose we have a functional equation of the form

$$f'(x) + g(x)f(x) = h(x) \tag{A.8}$$

that we would like to solve for $f(x)$, where $g(x)$ and $h(x)$ are continuous functions. Let $\mu(x)$ the *integrating factor*, be a function of x such that $\mu'(x) = \mu(x)g(x)$, rearranging this we got $\frac{\mu'(x)}{\mu(x)} = g(x)$. Hence

$$\log \mu(x) = \int g(x)dx + c \quad \text{hence} \quad \mu(x) = e^{\int g(x)dx + c} = e^c e^{\int g(x)dx} \tag{A.9}$$

Multiply both side of equation (A.8) by $\mu(x)$, we get

$$\begin{aligned}
\mu(x)f'(x) + g(x)\mu(x) &= h(x)\mu(x) \\
\mu(x)f'(x) + \mu'(x)f(x) &= h(x)\mu(x) \\
\big(f(x) + \mu(x)\big)' &= h(x)\mu(x) \\
\int \big(f(x) + \mu(x)\big)'dx &= \int h(x)\mu(x)dx \\
f(x) + \mu(x) + c' &= \int h(x)\mu(x)dx \\
f(x) &= \frac{\int h(x)\mu(x)dx + c'}{\mu(x)} = \frac{\int e^{\int g(x)dx} h(x)dx + C}{e^{\int g(x)dx}}
\end{aligned} \tag{A.10}$$

where $C = c'/e^c$ is the constant of integration.

Appendix B

Probability concentration theorems

B.1 The Chebyshev inequality

The theorem is named after Russian mathematician Pafnuty Chebyshev, although it was first formulated by his friend and colleague Irénée-Jules Bienaymé [32]. The following theorem statement is adapted from [19].

Theorem 13 (Chebyshev's inequality)**.** *Let X be a random variable with finite expected value μ and finite non-zero variance σ^2. Then for any real number $k > 0$,*

$$Pr(|X - \mu| \geq k\sigma) \leq \frac{1}{k^2}$$

Let $k\sigma = t$ thus $k = t/\sigma$, it follows that

$$Pr(|X - \mu| \geq t) \leq \frac{1}{(t/\sigma)^2} = \frac{\sigma^2}{t^2} = \frac{Var(X)}{t^2}$$

B.2 The Chernoff's bound

The following theorem statement is adapted from [41].

Theorem 14 (Chernoff's bound)**.** *Let $X = \sum_i^n X_i$ where X_i is a random variable such that $Pr(X_i = 1) = p$ and $Pr(X_i = 0) = 1 - p$ and all X_i are independent. Then*

$$Pr(|X - E[X]| \geq \delta E[X]) \leq 2e^{-\delta^2 E[X]/3} \quad (0 < \delta < 1) \tag{B.1}$$

Appendix C

Clustering metrics and algorithms

C.1 Quality metric of clustering

C.1.1 The modularity Q

The measure is calculated as follows [44]: let Q denote the modularity score, let A be a graph's adjacency matrix i.e. $A_{uv} = 0$ if there is no edge between vertex u and v, else $A_{uv} = 1$. Consider a graph $G(V,E)$, with $|V| = n$ and $|E| = m$, let $d(v)$ be the degree of a vertex v. Now, let us reconstruct G using the *configuration model*. Suppose for every vertex v its edges are cut in half leaving a number of $d(v)$ *stubs* and the total number of stubs is: $\sum_{v \in V} d(v) = 2m$. Then, the stubs are rewired by connecting a pair of stubs uniformly at random. As loops and multiple-edges may exist, we reject these cases and consider only simple graph.

Assuming vertices are belong in communities and the communities are indexed, let $c(v)$ denotes the community that vertex v belongs. Then define $\delta\{c(v), c(u)\}$ as a function that denotes the membership comparison between a pair vertex v and u which yields 1 if $c(v) = c(u)$ and 0 otherwise.

Select a pair of vertices v, u and let us calculate the probability that the edge (v,u) exists. For each stub of v, the probability that the stub connects with any of stub of u is: $d(u)/(2m-1)$. Since there are $d(v)$ stubs, the probability is

$$Pr\{(v,u)\} = \frac{d(v)d(u)}{2m-1} \approx \frac{d(v)d(u)}{2m}$$

for large m. The difference between actual number of edge and expected number of edge for that pair of vertices is then $A_{vu} - d(u)d(v)/2m$. The modularity is then defined as:

$$Q = \frac{1}{2m} \sum_{vu} \Big(A_{vu} - \frac{d(v)d(u)}{2m} \Big) \delta\{c(v), c(u)\} \tag{C.1}$$

Since $\delta = 0$ for every pair of vertices that does not belong in a cluster, the only contributors to the sum is therefore those belong in a same cluster. Let C be the set of clusters, then equation (C.1) can

be rewritten as:

$$Q = \sum_{c \in C} \Big(\sum_{vu} \frac{A_{vu}}{2m} - \sum_{u \in c} \frac{d(u)}{2m} \sum_{v \in c} \frac{d(v)}{2m} \Big) \tag{C.2}$$

Since the summations are taken on the adjacency matrix, the first summand counts twice the number of edges in c and the second sum is taken over every possible pair of vertices in c. Thus, we have the simple form of Q

$$Q = \sum_{c \in C} \Big(\frac{e_c}{m} - \Big(\frac{d_c}{2m} \Big)^2 \Big), \tag{C.3}$$

where e_c is the number of edges in cluster c and d_c is the sum degree of vertices in c.

C.1.2 Contingency table for pair-wise matching calculation

Consider two generic partitions $\mathscr{T}$ and $\mathscr{C}$ both contain a set of unique elements. Let $\mathscr{T} = \{T_1, T_2 \dots T_i\}$ ($i > 0$; $i \in N$), where each T_i is a *disjoint* set of vertices $T_i = \{u, v, \dots\}$; $\sum |T_i| = n$. And similarly, $\mathscr{C} = \{C_1, C_2 \dots C_j\}$, where j may or may not equal i; and each set $C_j = \{v, w, \dots\}$ is a *disjoint* set of vertices $\sum |C_j| = n$.

To compute the required pair:

1. a, the number of pairs of elements in V that are in the same set in $\mathscr{T}$ and in the same set in $\mathscr{C}$;
2. b, the number of pairs of elements in V that are in different sets in $\mathscr{T}$ and in different sets in $\mathscr{C}$;
3. c, the number of pairs of elements in V that are in the same set in $\mathscr{T}$ and in different sets in $\mathscr{C}$;
4. d, the number of pairs of elements in V that are in different sets in $\mathscr{T}$ and in the same set in $\mathscr{C}$;

consider following contingency table where each entry n_{ij} is the intersection between T_i and C_j:

	T_1	T_2	$\dots$	T_i	
C_1	n_{11}	n_{12}	$\dots$	n_{1i}	$n_{1.}$
C_2	n_{21}	n_{22}	$\dots$	n_{2i}	$n_{2.}$
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$
C_j	n_{r1}	n_{r2}	$\dots$	n_{ij}	$n_{j.}$
	$n_{.1}$	$n_{.2}$	$\dots$	$n_{.i}$	$n_{..} = n$

Table C.1 Contingency Table for Comparing Two Sets.

$n_{ij} = |T_i \cap C_j|$. Under the hypergeometric distribution Hubert [27] have shown that the expected index is:

$$E\Big[\sum_{i,j} \binom{n_{ij}}{2}\Big] = \Big[\sum_{i} \binom{n_{i.}}{2} \sum_{j} \binom{n_{.j}}{2}\Big] / \binom{n}{2}$$

Then ARI is given by:

$$f_{ARI} = \frac{\sum_{ij}\binom{n_{ij}}{2} - [\sum_i \binom{n_{i.}}{2}\sum_j \binom{n_{.j}}{2}]/\binom{n}{2}}{\frac{1}{2}[\sum_i \binom{n_{i.}}{2} + \sum_j \binom{n_{.j}}{2}] - [\sum_i \binom{n_{i.}}{2}\sum_j \binom{n_{.j}}{2}]/\binom{n}{2}} \tag{C.4}$$

Through some linear transformation, each individual element a, b, c, d can be calculated by:

- $a = \frac{1}{2}\sum_{ij} n_{ij}(n_{ij}-1)$
- $b = \frac{1}{2}\left(n^2 + \sum_{ij} n_{ij} - (\sum_i n_{i.}^2 + \sum_j n_{.j}^2)\right)$
- $c = \frac{1}{2}(\sum_j n_{.j}^2 - \sum_{ij} n_{ij})$
- $d = \frac{1}{2}(\sum_i n_{i.}^2 - \sum_{ij} n_{ij})$

C.2 Clustering algorithms in comparison

C.2.1 Edge-Betweenness Centrality Clustering.

The algorithm's procedure is as follows:

1. For each edge in the network, calculate its *edge-betweenness*;
2. Remove the edge with the highest betweenness;
3. Recalculate the edge-betweeness for all edges affected by the removal;
4. Repeat from step 2 until no edge remain.

It should be noted that, in [23] there is no direct instruction given regarding how to interpret the results obtained from the algorithm. Nevertheless, it is widely acknowledged that a *dendogram* is typically useful. In this direction, a dendogram can be built using the order of removal and the value of the *edge-betweennness* of the removed edges to indicate *height* of the *branch*. A procedure for building the dendogram can be as follows: assume the removed edges are stored in a stack (first-in, last-out buffer), pop the stack we obtain an edge that connects *exactly* a pair of leaf-vertices (vertex of degree 1), in the dendogram connect these pair by adding a *branch* with its height equals its *edge-betweenness*. Merge the pair and keep popping the stack, connect pair of vertices or *merged*-vertices until the stack is empty. The result dendogram is very useful to visually illustrate the arrangement of clusters i.e. in figure 7.5 the two large communities can be easily seen, connected by the top (highest) branch.

While being visually useful, the use of *dendogram* may requires some supervisions for interpreting results i.e. which *branch* to cut to obtain some number of desired-properties *communities*. Therefore, to compare the output with the *ground-truth*, we can either: stops when the number of communities

equals to a predefined number of communities k (number of communities in the ground-truth); or keeping track of the current Q at each iteration and take the *network-snapshot* that produce the highest Q as the output. Since the objective of the algorithm is to *maximise modularity*, we perform the *latter* in our experiments.

Naively, the *edge-betweenness clustering* is quite complex. In practice, to calculate the betweennesses a modified version of Breadth-First-Search is used, which takes $O(mn)$ for a graph of m edges and n vertices [42]. Since the procedure is repeated until no edge remains, the worst case running time of the algorithm is therefore $O(m^2n)$.

C.2.2 Fast Greedy Modularity Optimisation.

The main operations of this algorithm involves:

1. finding the changes in Q or ΔQ that would result from merging of each pair of communities;
2. choosing the largest increase ΔQ;
3. performing the merge.

Originally, in [43] these steps are carried out by imagine the original graph as a *multigraph*, in which a community is represented by a vertex, bundles together edges connect different communities, and intra-community edges are represented as self-edges. The adjacency matrix of this multigraph is then maintained after the joining of two communities i and j by replacing the i^{th} and j^{th} rows and columns by their sum; and so on. It is noted that calculating ΔQij and finding the pair i, j with the largest value then becomes time-consuming.

Clauset et al [8] point out that out that the update of the matrix involves a large number of useless operations, due to the sparsity of the adjacency matrix. Thus, this operation can be performed more efficiently by using appropriate data structures for sparse matrices, particularly:

- A sparse matrix containing ΔQ_{ij} for each pair i, j of communities with *at least one edge between them* (since two communities with no edge between them can never produce an increase in Q). Whence each row of the matrix is stored both as *a balanced binary tree* (so that elements can be found or inserted in O(log n) time) and as a maxheap (so that the largest element can be found in constant time).
- A max-heap containing the largest element of each row of the matrix ΔQ_{ij} along with the labels i, j of the corresponding pair of communities.
- An ordinary vector array with elements a_i.

Define

$$\Delta Q_{ij} = \begin{cases} e_{ij} - d(i)d(j)/(2m)^2 & \text{if } i \text{ and } j \text{ are connected;} \\ 0 \qquad \text{otherwise} \end{cases}$$

where e_{ij} is the fraction of edges that join vertices in community i and j; and $d(i)$ is the total degree of community i. And

$$a_i = \frac{d(i)}{2m}$$

which is the fraction of ends of edges that are attached to vertices in community i.

The procedure is then:

1. Calculate the initial values of ΔQ and a_i, and populate the max-heap with the largest element of each row of the matrix;
2. Select the largest ΔQ from max-heap, join the corresponding communities, update the matrix, max-heap and a_i and increment Q;
3. Repeat step 2 until only one community remains.

It is proved that the complexity of the algorithm is $O(md\log n)$, where d is the depth of the dendrogram, which grows as $\log n$ for graphs with a strong hierarchical structure, for which the running time is then $O(n\log^2 n)$.

C.2.3 Louvain Method for Large Networks

The *Louvain method*, consists of *two phases*, which are executed recursively:

1. Each *vertex* is considered a *community*. For each vertex v and its adjacency list adj_v; for each neighbour u of v: $u \in adj_v$ compute the gain in modularity ΔQ by putting v in community of u. Then, v *selects* the neighbour which yields the largest ΔQ, with the condition that $\Delta Q > 0$. Otherwise, v stays in its original community. The *first* phase stops when a local maxima is achieved.
2. Each community is merged as one *super-vertex*, and the process is repeated from step 1.

The end result of the algorithm yields a hierarchical, multi-level clustering in which the graph's structure corresponds to the number of time that step 2 is executed.

C.2.4 Infomap

The authors use *random walk* as the *proxy of information*, the process carries out as follows:

1. Each vertex in the graph is given a unique name using some efficient method of encoding i.e. Huffman encoding;
2. Let the *walk* takes some number k steps. For each step taken, its *traversed-vertex* is recored using its unique coding. The whole walk can then be represented by a sequence of vertices' name or *coding*.

3. Having the walk's *coding*, the problem of partitioning now can be translated to a *coding* problem: to find a partition that minimise the expected description length of the coding. The minimisation is achieved through incorporating greedy search with simulated annealing, more particularly the *modularity optimisation* strategy is adopted from *Louvain method*. To ensure ergodicity, a restart probability is later introduced.

References

[1] Infomap. http://www.mapequation.org/code.html. Accessed: 16/Sep/2016.

[2] LFR benchmark graph generator. https://sites.google.com/site/santofortunato/inthepress2. Accessed: 16/Sep/2016.

[3] Louvain. https://sites.google.com/site/findcommunities/. Accessed: 16/Sep/2016.

[4] Mathieu Bastian, Sebastien Heymann, and Mathieu Jacomy. Gephi: An open source software for exploring and manipulating networks, 2009. URL http://www.aaai.org/ocs/index.php/ICWSM/09/paper/view/154.

[5] Vincent D Blondel, Jean-Loup Guillaume, Renaud Lambiotte, and Etienne Lefebvre. Fast unfolding of communities in large networks. *Journal of Statistical Mechanics: Theory and Experiment*, 2008(10):P10008, 2008. URL http://stacks.iop.org/1742-5468/2008/i=10/a=P10008.

[6] Bela Bollobas. *Random Graphs*. Cambridge Studies in Advanced Mathematics. Cambridge University Press, 2 edition, 2001. doi: 10.1017/CBO9780511814068.

[7] Yizong Cheng. Mean shift, mode seeking, and clustering. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 17(8):790–799, Aug 1995. ISSN 0162-8828. doi: 10.1109/34.400568.

[8] Aaron Clauset, M. E. J. Newman, and Cristopher Moore. Finding community structure in very large networks. Phys. Rev. E 70, 066111 (2004), 2004.

[9] Anne Condon and Richard M. Karp. Algorithms for graph partitioning on the planted partition model. *Random Structures and Algorithms*, 18(2):116–140, 2001. ISSN 1098-2418. doi: 10.1002/1098-2418(200103)18:2<116::AID-RSA1001>3.0.CO;2-2. URL http://dx.doi.org/10.1002/1098-2418(200103)18:2<116::AID-RSA1001>3.0.CO;2-2.

[10] David J. Crandall, Lars Backstrom, Daniel Huttenlocher, and Jon Kleinberg. Mapping the world's photos. In *Proceedings of the 18th International Conference on World Wide Web*, WWW '09, pages 761–770, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-487-4. doi: 10.1145/1526709.1526812. URL http://doi.acm.org/10.1145/1526709.1526812.

[11] D. M. Mount and S. Arya. ANN: A Library for Approximate Nearest Neighbor Searching. https://www.cs.umd.edu/~mount/ANN/. Online; accessed 2017.

[12] Jesper Dall and Michael Christensen. Random geometric graphs. 2002. doi: 10.1103/PhysRevE.66.016121.

[13] Renato Cordeiro de Amorim and Christian Hennig. Recovering the number of clusters in data sets with noise features using feature rescaling factors. 2016. doi: 10.1016/j.ins.2015.06.039.

[14] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society. Series B (Methodological)*, 39(1):1–38, 1977. ISSN 00359246. URL http://www.jstor.org/stable/2984875.

[15] Carl P. Dettmann and Orestis Georgiou. Random geometric graphs with general connection functions. 2014. doi: 10.1103/PhysRevE.93.032313.

[16] Paul Erdős and Alfréd Rényi. On random graphs i. *Publicationes Mathematicae (Debrecen)*, 6: 290–297, 1959 1959.

[17] Martin Ester, HansPeter Kriegel, Jorg Sander, and Xiaowei Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. pages 226–231. AAAI Press, 1996.

[18] Vladimir Estivill-Castro. Why so many clustering algorithms: A position paper. *SIGKDD Explor. Newsl.*, 4(1):65–75, June 2002. ISSN 1931-0145. doi: 10.1145/568574.568575. URL http://doi.acm.org/10.1145/568574.568575.

[19] William Feller. *An Introduction to Probability Theory and its Applications Vol. I*. Wiley, 1968.

[20] Santo Fortunato. Community detection in graphs. *Physics Reports*, 486(3):75 – 174, 2010. ISSN 0370-1573. doi: http://dx.doi.org/10.1016/j.physrep.2009.11.002. URL http://www.sciencedirect.com/science/article/pii/S0370157309002841.

[21] Linton C. Freeman. A set of measures of centrality based on betweenness. *Sociometry*, 40(1): 35–41, 1977. ISSN 00380431. URL http://www.jstor.org/stable/3033543.

[22] Alan Frieze and Karonski Michal. *Introduction to Random Graphs*. Cambridge University Press, 2015. doi: 10.1017/CBO9781316339831.

[23] Michelle Girvan and M. E. J. Newman. Community structure in social and biological networks. Proc. Natl. Acad. Sci. USA 99, 7821-7826 (2002), 2001.

[24] David Goodmanson and Eric W. Weisstein. "laurent series". MathWorld–A Wolfram Web Resource. URL http://mathworld.wolfram.com/LaurentSeries.html.

[25] Piyush Gupta and P. R. Kumar. *Critical Power for Asymptotic Connectivity in Wireless Networks*, pages 547–566. Birkhäuser Boston, Boston, MA, 1999. ISBN 978-1-4612-1784-8. doi: 10.1007/978-1-4612-1784-8_33. URL https://doi.org/10.1007/978-1-4612-1784-8_33.

[26] Kriegel Hans-Peter, Kröger Peer, Sander Jörg, and Zimek Arthur. Density-based clustering. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 1(3):231–240. doi: 10.1002/widm.30. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/widm.30.

[27] Lawrence Hubert and Phipps Arabie. Comparing partitions. *Journal of Classification*, 2(1): 193–218, 1985. ISSN 1432-1343. doi: 10.1007/BF01908075. URL http://dx.doi.org/10.1007/BF01908075.

[28] Paul Jaccard. Etude de la distribution florale dans une portion des alpes et du jura. *Bulletin de la Societe Vaudoise des Sciences Naturelles*, 37:547–579, 01 1901.

[29] Svante Janson. Simply generated trees, conditioned galton–watson trees, random allocations and condensation. arXiv:1112.0510, 2011.

[30] data science KDD: The community for data mining and analytics. Sigkdd test of time award. Online, August 2014.

[31] K. Knopp. *Theory of Functions Parts I and II, Two Volumes Bound as One*, volume I. New York: Dover, 1996.

[32] Donald E. Knuth. *The Art of Computer Programming, Volume 1 (3rd Ed.): Fundamental Algorithms*. Addison Wesley Longman Publishing Co., Inc., Redwood City, CA, USA, 1997. ISBN 0-201-89683-4.

[33] Hans-Peter Kriegel, Peer Kröger, Erich Schubert, and Arthur Zimek. Loop: Local outlier probabilities. In *Proceedings of the 18th ACM Conference on Information and Knowledge Management*, CIKM '09, pages 1649–1652, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-512-3. doi: 10.1145/1645953.1646195. URL http://doi.acm.org/10.1145/1645953.1646195.

[34] Andrea Lancichinetti, Santo Fortunato, and Filippo Radicchi. Benchmark graphs for testing community detection algorithms. Physical Review E78, 046110 (2008), 2008.

[35] Jure Leskovec and Rok Sosič. SNAP: A general-purpose network analysis and graph-mining library. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 8(1):1, 2016.

[36] Hartmut Logemann and Eugene P. Ryan. *Ordinary differential equations. Analysis, qualitative theory and control*. 01 2014.

[37] David Lusseau and M. E. J. Newman. Identifying the role that animals play in their social networks. *Proc Biol Sci*, 271(Suppl 6):S477–S481, Dec 2004. ISSN 0962-8452. URL http://www.ncbi.nlm.nih.gov/pmc/articles/PMC1810112/. 15801609[pmid].

[38] David Lusseau, Karsten Schneider, Oliver J. Boisseau, Patti Haase, Elisabeth Slooten, and Steve M. Dawson. The bottlenose dolphin community of doubtful sound features a large proportion of long-lasting associations. *Behavioral Ecology and Sociobiology*, 54(4):396–405, Sep 2003. ISSN 1432-0762. doi: 10.1007/s00265-003-0651-y. URL https://doi.org/10.1007/s00265-003-0651-y.

[39] M. Ciollaro and D. Wang. Package: MeanShift. https://cran.r-project.org/web/packages/MeanShift/MeanShift.pdf. Online; accessed 2017.

[40] M. Hahsler et al. Package: dbscan. https://cran.r-project.org/web/packages/dbscan/dbscan.pdf. Online; accessed 2017.

[41] Michael Mitzenmacher and Eli Upfal. *Probability and Computing: Randomized Algorithms and Probabilistic Analysis*. Cambridge University Press, New York, NY, USA, 2005. ISBN 0521835402.

[42] M. E. J. Newman. Scientific collaboration networks. ii. shortest paths, weighted networks, and centrality. *Phys. Rev. E*, 64:016132, Jun 2001. doi: 10.1103/PhysRevE.64.016132. URL http://link.aps.org/doi/10.1103/PhysRevE.64.016132.

[43] M. E. J. Newman. Fast algorithm for detecting community structure in networks. *Phys. Rev. E 69, 066133 (2004)*, 2004. doi: 10.1103/PhysRevE.69.066133.

[44] M. E. J. Newman. Modularity and community structure in networks. *Proceedings of the National Academy of Sciences*, 103(23):8577–8582, 2006. ISSN 0027-8424. doi: 10.1073/pnas.0601602103. URL http://www.pnas.org/content/103/23/8577.

[45] M. E. J. Newman and M. Girvan. Finding and evaluating community structure in networks. Phys. Rev. E 69, 026113 (2004), 2004.

[46] Mark Newman. *Networks: An Introduction*. Oxford University Press, Inc., New York, NY, USA, 2010. ISBN 0199206651, 9780199206650.

[47] Stuart P. Lloyd. Least squares quantization in pcm's. 28:129–136, 03 1982.

[48] M. Penrose. *Random Geometric Graphs*. Oxford studies in probability. Oxford University Press, 2003. ISBN 9780198506263. URL https://books.google.co.uk/books?id=M38e7nPGSCsC.

[49] William M. Rand. Objective criteria for the evaluation of clustering methods. *Journal of the American Statistical Association*, 66(336):846–850, 1971. ISSN 01621459. URL http://www.jstor.org/stable/2284239.

[50] Martin Rosvall and Carl T. Bergstrom. Maps of random walks on complex networks reveal community structure. *Proceedings of the National Academy of Sciences*, 105(4):1118–1123, 2008. doi: 10.1073/pnas.0706851105. URL http://www.pnas.org/content/105/4/1118.abstract.

[51] Peter J. Rousseeuw. Silhouettes: A graphical aid to the interpretation and validation of cluster analysis. *Journal of Computational and Applied Mathematics*, 20(Supplement C):53 – 65, 1987. ISSN 0377-0427. doi: https://doi.org/10.1016/0377-0427(87)90125-7. URL http://www.sciencedirect.com/science/article/pii/0377042787901257.

[52] Thomas Schank and Dorothea Wagner. *Finding, Counting and Listing All Triangles in Large Graphs, an Experimental Study*, pages 606–609. Springer Berlin Heidelberg, Berlin, Heidelberg, 2005. ISBN 978-3-540-32078-4. doi: 10.1007/11427186_54. URL https://doi.org/10.1007/11427186_54.

[53] Bubeck Sébastien, Ding Jian, Eldan Ronen, and Rácz Miklós Z. Testing for high-dimensional geometry in random graphs. *Random Structures & Algorithms*, 49(3):503–532. doi: 10.1002/rsa.20633. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/rsa.20633.

[54] Steven S. Skiena. *The Algorithm Design Manual*. Springer Publishing Company, Incorporated, 2nd edition, 2008. ISBN 1848000693, 9781848000698.

[55] Brian S. EverittSabine LandauMorven LeeseDaniel Stahl Brian S. EverittSabine LandauMorven LeeseDaniel Stahl Brian S. EverittSabine LandauMorven LeeseDaniel Stahl Brian S. EverittSabine LandauMorven LeeseDaniel Stahl. *Cluster Analysis*. Number ISBN:9780470749913 in Wiley Series in Probability and Statistics. John Wiley and Sons, Ltd, January 2011.

[56] Elias M. Stein and Rami Shakarchi. *Complex Analysis*, volume 2nd. Princeton University Press, April 27, 2003.

[57] Various. Boost.Geometry. http://www.boost.org/doc/libs/1_61_0/libs/geometry/doc/html/index.html. Online; accessed 2017.

[58] Duncan J. Watts and Steven H. Strogatz. Collective dynamics of /'small-world/' networks. *Nature*, 393(6684):440–442, Jun 1998. ISSN 0028-0836. doi: 10.1038/30918. URL http://dx.doi.org/10.1038/30918.

[59] Duncan J. Watts and Steven H. Strogatz. Collective dynamics of 'small-world'networks. *Nature*, 393:440 EP –, 06 1998. URL http://dx.doi.org/10.1038/30918.

[60] Eric Weisstein. Disk point picking. MathWorld–A Wolfram Web Resource. http://mathworld.wolfram.com/DiskPointPicking.html, 2003.

[61] Eric W Weisstein. First-order ordinary differential equation. From MathWorld–A Wolfram Web Resource. URL http://mathworld.wolfram.com/First-OrderOrdinaryDifferentialEquation.html.

[62] Wikipedia. DBSCAN. https://en.wikipedia.org/wiki/DBSCAN. Online; accessed 2017.

[63] Herbert S. Wilf. *Generatingfunctionology*. A. K. Peters, Ltd., Natick, MA, USA, 1990. ISBN 1568812795.

[64] Wayne W. Zachary. An information flow model for conflict and fission in small groups. *Journal of Anthropological Research*, 33(4):452–473, 1977. ISSN 00917710. URL http://www.jstor.org/stable/3629752. Essential.